



SUDAN UNIVERSITY OF SCIENCE AND TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
COMPUTER SCIENCE AND INFORMATION SYSTEM DEPARTMENT

**COMPARING THE PERFORMANCE OF APACHE
SPARK AND APACHE HADOOP MAPREDUCE
ON BIG DATA PROCESSING**

THESIS SUMMITTED AS A PARTIAL FULFILLMENT OF B.Sc. (HONOR) DEGREE
IN COMPUTER SCIENCE AND INFORMATION SYSTEM

OCTOBER 2017

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

**SUDAN UNIVERSITY OF SCIENCE AND
TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND
INFORMATION TECHNOLOGY
COMPUTER SCIENCE AND INFORMATION SYSTEM
DEPARTMENT**

**COMPARING THE PERFORMANCE OF APACHE
SPARK AND APACHE HADOOP MAPREDUCE ON BIG
DATA PROCESSING**

PREPARED BY:

ALAA ISMAIEL IBRAHIM SHUMO.

ESRA ADIL GALAL SALIH.

SAJDA LOTFY AHMED KHALED.

SARA HASSABO ABDALLAH ALBASHEER.

SUPERVISOR:

AHMED HAMZA ABDL-MONIEM HAMZA

SIGNATURE OF SUPERVISOR:

.....

DATE:

.....

الآية

قال تبارك وتعالى:

﴿وَلَقَدْ كَرَّمْنَا بَنِي آدَمَ وَحَمَلْنَاهُمْ فِي الْبَرِّ وَالْبَحْرِ وَرَزَقْنَاهُمْ مِنَ

الطَّيِّبَاتِ وَفَضَّلْنَاهُمْ عَلَى كَثِيرٍ مِمَّنْ خَلَقْنَا تَفْضِيلًا﴾

(الإسراء - 70)

الحمد لله

الحمد لله عالم السر والجهر، وقاصم الجبابرة بالعز والقهر، مُحصي قطرات الماء وهو يجري في النهر، فضَّل بعض المخلوقات على بعض حتى أوقات الدهر، فهو المتفرد بإيجاد خَلْقِهِ، المتوحد بإدرا رزقه .

القديم فالسَّبْق لسبقه، الكريم فما قام مَخْلُوقٌ بحقه، عالمٌ بسر العبد وسامعٌ نطقه، ومقدر علمه وعمله وعمره وفعله وخلقه، ومجازيه على عيبه وذنبه وكذبه وصدقه، المالك القَهَّار فالكلُّ في أسر رقه، الحليم الستار فالخلق في ظل رفقهِ، أرسل السحاب تخُاف صواعقه ويَطْمَع في وَدْقِهِ، يزعج القلوب رواعده، ويكاد سنا بَرَقِهِ، جعل الشمس سراجًا والقمر نورًا بيِّنَ غَرْبِهِ وشرْقِهِ .

نَحْمده على الهدى وتسهيل طرقه، وأشهد أن لا إله إلا الله وحده لا شريك له في رَتَقِهِ وفتَقِهِ، وأنَّ مُحَمَّدًا عبدهُ ورسوله، أرسله والضلال عامٌّ فمحاها بِمَحَقِهِ، صلى الله عليه وسلم وعلى آله وصحابه أبي بكر السابق ب صدقته، وعلى عمر كاسر كَسْرِي بتدبيره و حدْقِهِ، وعلى عثمان جامع القرآن بعد تبديده في رَقِهِ، وعلى عليٍّ واعذرونا في عشقته، وعلى عمه العباس مشاركةً في أصله و عزْقِهِ .

الإهداء

إلى من نذرت عمرها في أداء رسالة صنعتها من أوراق الصبر وطرزتها في ظلام الدهر
رسالة تعلم العطاء كيف يكون العطاء وتعلم الوفاء كيف يكون الوفاء إليك

"أمي"

أهدي هذه الرسالة وشتان بين رسالة ورسالة جزاك الله خيراً.. وأمد في عمرك بالصالحات
فأنت زهرة الحياة ونورها.

إلى من جرع الكأس فارغاً ليسقيني قطرة حب إلى من كلت أنامله ليقدم لنا لحظة سعادة إلى
من حصد الأشواك عن دربي ليمهد لي طريق العلم إلى القلب الكبير.

"أبي"

إلى من كانوا يضيئون لي الطريق ويساندوني ويتنازلون عن حقوقهم لإرضائي والعيش في
هناء، أحبكم حبا لو مر على أرض قاحلة لتفجرت منها ينابيع المحبة.

"صديقاتي ورفيقاتي في الحياه"

إلى كل من ساندنا بالكلام او الفعل من قريب او البعيد.

إلى كل طلبة السنة الرابعة تخصص نظم المعلومات والشبكات دفعة 2013

إلى كل من سقط من قلبي سهوا

أهدي هذا العمل

شكر و عرفان

الحمد لله رب العالمين، والصلاة والسلام على أشرف الخلق والمرسلين نبينا محمد صلى الله عليه وسلم وعلى آله الطيبين الطاهرين.

أول الشكر وآخره نتقدم به إلى المنعم الباري عز وجل (الله) سبحانه وتعالى، الذي أحاطنا برعايته الإلهية العظيمة، ويسر لنا كل عسير، وألهمنا الصبر والقوة في شق طريقنا نحو البحث العلمي. نتقدم بخالص الشكر الجزيل والعرفان بالجميل والاحترام والتقدير لمن غمرنا بالفضل واختصنا بالنصح للأستاذ المشرف/أحمد حمزه، فقد كان قيس الضياء في عتمة البحث، كما كان قبطان مركب العلم في هوج الدراسة المتلاطم ولعلنا لا نعدو الحق إذ نقول أنه كان لنا نعم الناصح الأمين ونعم الأب الوقور ونعم الأخ الحلیم أفاض علينا بعلمه وشمأننا بفضلہ وسماحتہ، منحنا الثقة وغرس في نفوسنا قوة العزيمة ولم يدخر جهداً، ولم يبخل علينا بأبوابه الله ذخراً لطلبة العلم وجعل ذلك في ميزان حسناته وأرضاه بما قسم له. ولا ننسى أن نتقدم بجزيل الشكر للمهندس/محمد عبدالرحيم الذي قام بتوجيهنا طيلة هذه الدراسة.

وفاءً وتقديراً وإعترافاً منا بالجميل نتقدم بجزيل الشكر لكل الاساتذه المخلصين الذين لم يألوا جهداً في مساعدتنا في مجال البحث العلمي، والذين كان لهم الفضل في توجيهنا ومساعدتنا في جميع المادة البحثية، ونخص بالشكر والعرفان: د. هويده على عبد القادر، أ. هشام، أ.وفاء فيصل، أ.نفيسه وأ.غاده على شمو، فجزاهم الله كل خير. وأخيراً، نتقدم بجزيل شكرنا إلى كل من مدوا لنا يد العون والمساعدة في إخراج هذه الدراسة علي أكمل وجه.

إلى انفسنا التي لاقت من الصعوبات ما لاقت، والتي لم تتخطاها سوى بحسن الظن وقوه التوكل على الله وحده.

Abstract

Imagine the massive volume of data in the world, and the rapid growth of it every moment and every second, these data that carry many useful values, which help companies to succeed and increase a competitive advantage, is called 'Big Data', due to its sheer Volume, Variety, Velocity and Veracity. Most of this data is unstructured, structured or semi structured.

The large amounts of data created a need for new frameworks for processing. The “*Apache Hadoop MapReduce*” model is a framework for processing large-scale datasets with parallel and distributed algorithms. The “*Apache Hadoop MapReduce*” allows for the distributed processing of large data sets across clusters of computers using simple programming models.

Recently a framework called Apache Spark has emerged, focused on micro-batch data processing. In addition the main feature of Spark is the in-memory computation.

In this research, we perform a comparative study on the performance of these two frameworks. Additionally we use bigdatabench (tool) to load dataset up to 420 million records. Experimental results show that Spark has better performance and overall lower runtimes than Apache Hadoop MapReduce.

المستخلص

تخيل ذلك الحجم الهائل من البيانات في العالم، والنمو السريع في كل لحظة وكل ثانية، وأن هذه البيانات تحمل العديد من القيم المفيدة، والتي تساعد الشركات على النجاح وزيادة ميزتها التنافسية، هذه البيانات نطلق عليها مصطلح "البيانات الكبيرة"، نظراً لحجمها الهائل، وتنوعها وسرعتها ومدى صحتها، ومعظم هذه البيانات مهيكلة أو شبه مهيكلة أو غير مهيكلة.

لذلك وجدت الحاجة الماسة إلى ظهور أطر عمل جديدة لمعالجة تلك البيانات الكبيرة، منها إطار Hadoop MapReduce وهو إطار يستخدم الخوارزميات الموزعة والمتوازية لمعالجة البيانات الكبيرة على المدى الواسع، وإيضاً يسمح بتوزيع المعالجة على البيانات لمجموعه من أجهزة الكمبيوتر (clusters) باستخدام نماذج برمجة بسيطة.

في الأونة الأخيرة برز إطار يسمى Apache Spark ، ركز على micro-batch data

processing. بالإضافة إلى ان الميزة الرئيسية ل Apache Spark هي المعالجة في الذاكرة.

في هذا البحث، قمنا بإجراء مقارنة عن أداء هذين الإطارين. بالإضافة إلى ذلك استخدمنا

BigDataBench لتحميل بيانات تصل إلى 420 مليون سجل. وأظهرت النتائج التجريبية أن Apache

Spark كان أداءه أفضل، و زمن تنفيذه أقل من Hadoop MapReduce.

Table of Contents

CHAPTER ONE: INTRODUCTION	1
1.1 INTRODUCTION	2
1.2 PROBLEM STATEMENT	3
1.3 OBJECTIVES	3
1.4 RESEARCH SIGNIFICANCE	4
1.5 PROPOSED SOLUTION	4
1.8 THESIS STRUCTURE	5
CHAPTER TWO: THEORETICAL BACKGROUND	7
2.1 INTRODUCTION	8
2.2 DETAILS ABOUT BIG DATA SCIENCE	8
2.3 HISTORICAL BACKGROUND	9
2.4 BIG DATA PROCESSING TYPES	10
2.4.1 BATCH PROCESSING	10
2.4.2 REAL-TIME DATA PROCESSING	12
2.4.2.1 IN-MEMOREY COMPUTING	12
2.4.2.2 REAL-TIME QUERIES OVER BIG DATA	13
2.4.3 STREAMING BIG DATA	13
2.5 OTHER BIG DATA FRAMEWORKS	15
2.5.1 APACHE STORM	15
2.5.2 APACHE FLINK	15
2.6 SUMMARY	16

CHAPTER THREE: LITERATURE REVIEW	17
3.1 INTRODUCTION	18
3.2 PREVIOUS STUDIES	18
3.2.1 COMPARISON BETWEEN FRAMEWORKS IN PERFORMANCE	18
3.2.2 PRPROCESSING	20
3.2.3 BENCHMARKING	21
3.3 SUMMARY	24
CHAPTER FOUR: TOOLS, TECHNIQUES AND RESEARCH METHODOLOGY	25
4.1 INTRODUCTION	26
4.2 TOOLS AND TECHNIQUES	26
4.2.1 APACHE HADOOP	26
4.2.2 APACHE SPARK	28
4.2.3 APACHE HIVE	29
4.2.4 BIG DATABENCH	30
4.3 RESEARCH METHODOLOGY	31
4.3.1 SURVEY	31
4.3.1.1 SURVEY PURPOSES	31
4.3.1.2 SAMPLE DISTRIBUTION AND FILLING UP	31
4.3.1.3 SURVEY QUESTIONS	32
4.3.1.4 SURVEY RESULTS	32
4.3.2 PREPARE THE ENVIROMENT	39
4.3.3 CONFIGURES CONNECTION BETWEEN MACHINES	40

4.3.3.1 CONFIGURING “HADOOP” AND “SPARK” AND “HIVE” IN ALL MACHINES	41
4.3.3.2 DATA GENERATION	43
4.3.4 RUNNING JOBS	46
4.3.4.1 Run Word-Count Workload on Hadoop cluster consist of one NameNode and two DataNodes	46
4.3.4.2 Run “select” query on The E- Commerce and Spark cluster and Spark cluster consist of one NameNode and 3 DataNodes	48
4.4 SUMMARY	53
CHAPTER FIVE: RESULTS AND RECOMMENDATIONS	54
5.1 INTRODUCTION	55
5.2 RESULTS	55
5.2.1 PERFORMANCE RESULTS OF “WORD-COUNT” JOB ON HADOOP CLUSTER	55
5.2.2 PERFORMANCE MEASUREMENTS RESULTS OF “select” QUERY IN HADOOP CLUSTER	56
5.2.3 RESULTS OF “select” QUERY IN SPARK CLUSTER	57
5.2.4 COMPARISON RESULT BASED ON TIME OF PROCESSING	58
5.3 RECOMMENDATIONS	60
References	61
APPENDICES	64
APPENDIX (A)	65

CONFIGURATION OF FRAMEWORKS	65
APPINDEX (B)	88
BIG DATA IN SUDAN QUESTIONNAIRE	88

List of Figures

Figure 2.1: illustrate processing in Spark	15
Figure 4.1: Architecture of MapReduce execution.....	28
Figure 4.2: illustrate Question 1.....	32
Figure 4.3: illustrate Question 2.....	33
Figure 4.4: illustrate Question 3.....	33
Figure 4.5: illustrate Question 4.....	34
Figure 4.6: illustrate Question 5.....	34
Figure 4.7: illustrate Question 6.....	35
Figure 4.8: illustrate Question 7.....	35
Figure 4.9: illustrate Question 8.....	36
Figure 4.10: illustrate Question 9	36
Figure 4.11: illustrate Question 10	37
Figure 4.12: illustrate Question 11	37
Figure 4.13: illustrate Question 12	38
Figure 4.14: illustrate Question 13	38
Figure 4.15: SSH steps	40
Figure 4.16: format NameNode.....	41
Figure 4.17: Successfully formatted.....	42
Figure 4.18: Starting Hadoop daemon	43
Figure 4.19: generated 30-gigabyte data	44
Figure 4.20: ORDERS.txt file.....	45
Figure 4.21: ORDERS _ITEM.txt file	46
Figure 4.22: Run_Microbenchmarks.sh file	47
Figure 4.23: start JobHistory service	48
Figure 4.24: JobHistory UI	48
Figure 4.25: create a folder in HDFS called Hive.....	49

Figure 4.26: created table in hive	49
Figure 4.27: Select Query in hive	50
Figure 4.28: MapReduce started doing the select job	50
Figure 4.29: Start Spark shell.....	51
Figure 4.30: write queries and interact with hive “metastore” using “HiveQL”	51
Figure 4.31: Spark context value	52
Figure 4.32: load the data stored in “ORDERS_ITEM.txt”	52
Figure 4.33: store the result on “resu” variable	52
Figure 5.1: measurements was saved in “JobHistory”	56
Figure 5.2: result measurements in “SparkJobs”	58

List of Tables

Table 1: Table 3.1 illustrate our opinion.....	24
Table 2: Table 5.1 illustrate comparison result.....	58

List of Abbreviations

#	Term	Description
1	BDGS	Big Data Generate Suite
2	HDFS	Hadoop Distributed File System
3	BI	Business Intelligence
4	RDD	Resilient Distributed Dataset
5	ETL	Extract, Transform, Load
6	RDBMS	Relational Database Systems
7	GFS	Google File System
8	DStream	Discretized Stream
9	JVM	Java Virtual Machine

CHAPTER ONE

INTRODUCTION

1.1 INTRODUCTION

In 2010 year the 'Big Data' was virtually unknown, but by mid-2011 it was being diffuse widely as the hot trend, the term has today been adopted by everyone, from product vendors to large-scale outsourcing and cloud service providers intensive to promote their offerings. But what really is Big Data?

Big Data is about quickly deriving business value from a range of new and emerging of new technologies, devices and communication means like social network sites, which led to a noticeable increase of the amount of data produced every year, even every day. In addition, traditional algorithms and technologies are inefficient to process, analyze and store this vast amount of data. [1]

And much more besides we can defining Big Data by the 3V models) volume, velocity, variety), then, with the development of the large data, a new feature called "veracity" was added, which is then called 4 V's of big data. Which we will explain each of these characteristics by the next chapters. Also Big Data is about how these data can be stored, processed, and comprehended such that it can be used for predicting the future course of action with a great precision and acceptable time delay.

On the other hand big data has a lot of challenges, so we want to touch these challenges of big data, and try to enumerate some of these challenges, but not limited to. One of these challenges is storage and retrieval of vast amount of structured as well as unstructured data which leads to time lag, another challenge is regarding to handle and process vast amount of data with the traditional storage techniques, these challenges is the main reason that led to emergence the term Big Data[2]. So these challenges will require treated solutions, we must support and encourage fundamental research towards addressing these technical challenges if we are to achieve the promised benefits of Big Data [3].

In this research we need to address the challenge of data processing on Big Data, data processing is common part of processes inside every organization. Critical challenges of these days came with Big Data processing. Although new technologies appeared, traditional data sources and processes require variety of different approaches [4].

In this research, we focus on studying two kind of very popular and most used frameworks in Big Data field, they are Hadoop MapReduce and Spark. To illustrate how these frameworks can service the data processing in well format. When it comes to processing Big Data, Hadoop MapReduce and Spark must be the first choices, but they aren't the only options. (Hadoop MapReduce) is actually quite simple. If your data can be processed in batch, split into smaller processing jobs, spread across a cluster, and their efforts recombined, hadoop will probably work just fine for you.

Spark differs from Hadoop MapReduce in that it works in-memory, speeding up processing times [5].

1.2 PROBLEM STATEMENT

We take performance on data processing over Big Data as a problem on the one hand, data processing is common part of processes and activities inside every organization. “Critical challenges on these days came with Big Data processing.

On the other hand, we have chosen two big data solution frameworks, they are Hadoop MapReduce and Spark, and saw the problem of how to choose one of them, based on company’s resources, company’s needs, and company’s big data.

1.3 OBJECTIVES

- Implement a “Word-Count” Example on Unstructured data set using Hadoop MapReduce, and get the performance measurements, consist of: time performance, CPU spent time, RAM consumptions.
- Execute a “test” query on structural big data set using Hadoop MapReduce and Spark, to compare between them considering the time performance and error handling.
- Find the cases which Hadoop MpaReduce is the best than Spark and vice versa.
- Help company’s decision makers (e.g. top manager) to know the better big data solution (Hadoop or Spark) that should use according to his needs.

1.4 RESEARCH SIGNIFICANCE

Big data has many processing solutions, this solutions have been already built by large companies like Apache. Because companies which have big data or unfortunately suffering from big data processing problem, big data frameworks are needed.

But in Sudan these solutions are not implemented yet (according to the survey was done in the research).

Decision-maker of these companies can choose solution that requires very high capacity of rams or disks and pay a lot of money, although there is no need to pay all that money! Because another solution is suitable for him and solve his problem without big lose, so decision-maker must know all solutions of big data, and this is very hardly job. He requires an implemented comparison between the big data solutions and clean road map to make his decision.

Reports and queries and business intelligence are built over data processing, so when data processing takes a lot of time; it is also cause a delay in production time of reports and queries, so this problem leads to time and money consuming. e.g. When there is company (x) started to generate a report that will be useful to it, and it uses a traditional database techniques to generate this report, and there is a competitor company (y) generates this report using big data framework before company (x); it will be useless report for company (x) and it consumes its resources on the air.

1.5 PROPOSED SOLUTION

This research will focus on two big data processing frameworks to compare between them based on three criteria's: time performance, configuration method, and error and exception handling, they are Hadoop MapReduce and Spark, understand functionality and use of each framework, how and where they are store and process data, and so on.

Then will execute some work on each framework, and then obtain the resources consumptions of machines when running this work on the same data set and on the same cluster using each framework, after that a simple comparison will be made between these two frameworks based on time performance and implementing way and clearness of error and exceptions on each framework.

1.6 RESEARCH SCOPE

The scope of this research is implementation and comparison of two processing frameworks Hadoop MapReduce and Spark, and their role in processing data. This comparison will based on time performance of query, the configuration way of two framework, and error and exception handling.

Also the scope will include measure CPU time spent, memory usage and running time of the Hadoop cluster, but not for Spark.

1.7 RESEARCH METHODOLOGY

The methodology for this research will be include:

- Work survey to find out the importance of this research and causes of needing big data frameworks.
- Generate unstructured data from “BigDataBench” suite as text data set, contains Wikipedia entries on search engine and this data is will be uploaded on Hadoop file system to store it there.
- Working model “workload” called “Word Count” will be executed on this data by Hadoop MapReduce framework.
- When the workload has been completed the performance criteria “time performance, CPU time spent and RAM usage” must be computed.
- This research will address generation of another type of data, it is structured data contains 420,000,000 record, and a simple query job will be implemented using Hadoop MapReduce and Spark frameworks, after that, a comparison will be done between these two frameworks (Hadoop MapReduce and Spark) based on configuration methods and errors handling and time performance of the query.

1.8 THESIS STRUCTURE

In addition to this chapter this research contains another four chapters:

- Chapter two: include the theoretical Background about sciences (big data and its frameworks and also big data processing).

- Chapter three: include literature review and related works.
- Chapter four: include the tools and techniques that used in our research, and also include the methodology and the activities that we did in our project implementation.
- Chapter five: include results and recommendations.

CHAPTER TWO

THEORETICAL BACKGROUND

2.1 INTRODUCTION

Since big data has its own characteristics such as size, diversity and speed of growth, it makes it difficult to process and manage them in traditional ways such as relational database management system. In this chapter we talk about:

- Details about Big Data science.
- Some big data processing types.
- Some frameworks of big data.

2.2 DETAILS ABOUT BIG DATA SCIENCE

The term big data is refers to: "datasets which have size that outside the capabilities of traditional database software tools to capture, store, manage, and analyze". [4] It can be structured, semi structured, and unstructured data based on context.

There are four key properties that define big data:

- **Volume:** The volume of data indicates to the size of data which controlled by the system. Data which is to some degree habitually generated tends to be big. For example data which generated by sensors in manufacturing or processing plants, data which generates from scanning equipment looks like smart and credit card readers, also data from measurement devices like soundtrack devices and the data which generated from the internet of things is huge data because these billion of devices which connected to the Internet generate data constantly.

- **Velocity:** It's about the speed at which data is generated, collected, ingested, and processed. High velocity is attributed to data when we consider the typical speed of transactions on normal exchanges; this speed touches billions of transactions per day on certain days. For example in twitter Velocity is useful in discovering trends among people that are making million tweets every three minutes.
- **Variety:** Data is generated from different types of sources and these sources have extended and contain for example Internet data like social media, research data surveys, location data like mobile device data, images like satellites and video data YouTube inserts hundreds of minutes of video every minute Big data contains many kinds of data first, structured data are in the form of tables containing rows and columns second, and semi structured data which data doesn't all have to track a static predefined structure. The third kind is unstructured data which haven't recognizable formal construction this kind of data establishes the main challenge in today's big data systems.
- **Veracity:** Veracity has two fixed features: the reliability of the source, and the appropriateness of data for its target listeners. [6]

2.3 HISTORICAL BACKGROUND

Big data can cause a huge problem for large companies when this company doesn't make use of it. Google which is the most widely used search engine has the first idea to solve big data problems, and gathers huge amount of data every day. Google find out two key technologies to handle this amount of data they want to store and analyze it.

Google first established Google File System which is a distributed storage model, and became the underlining storage architecture for the large volume of data which need to store. GFS runs on a large array of cheap hardware. The paper of Google File System is published in 2003 by Google, after that in 2004, they published extra paper on their distributed computing system called MapReduce.

Hadoop was made as an open source version as a result of these two basic technologies from Google the google technologies. And we will talk about it in chapter 4. In 2009 Additional framework called Spark was first established in the” AMPLab”, and also we will talk about it in chapter 4. [7]

2.4 BIG DATA PROCESSING TYPES

This section described these popular types of data processing: Batch processing, real-time data processing and Streaming Big Data.

2.4.1 BATCH PROCESSING

Batch processing is a technique that enables processes data in one large group, instead of individually groups. MapReduce is one of famous solution model that using big data batch processing. It is introduced and used by Google. [8]

MapReduce has three major Characteristics in its single package. These Characteristics are: simple and easy programming model, automatic and linear scalability, and built-in fault tolerance.

Google published its MapReduce framework with three major components: a MapReduce execution engine, distributed file system called GFS, and a distributed NoSQL database called BigTable. After Google’s published its MapReduce Framework, Apache foundation started some open source execution projects on MapReduce framework. Such as: [9]

- HDFS: is a distributed file system which delivers high-throughput access to application data.
- Hadoop YARN: is a framework used to schedule the jobs and cluster resource management.

- Hadoop MapReduce: is a system based on YARN for parallel processing of large data sets. [10]

The MapReduce framework has made complex large-scale data processing simple and efficient. From this despite MapReduce is designed for batch processing of large volumes of data, and it is not fit for recent demands like real-time and online processing. MapReduce is considered for high throughput batch processing of big data that take several hours and even days.

There are many systems which are implemented the distributed system via the MapReduce method like: Apache Hadoop, Disco from Nokia, HPC from LexisNexis, Dryad from Microsoft, and Sector/Sphere. However, Hadoop is the most well-known and popular open source implementation of MapReduce.

Apache Hadoop is one of the big data open source frameworks which implemented the distributed system by using the MapReduce techniques. Apache has many projects which related to Hadoop and these projects are:

- **Ambari**: is a tool based on web which used for provisioning, managing, and observing Apache Hadoop clusters which includes support for Hadoop HDFS, Hadoop MapReduce, Hive, HCatalog, HBase, ZooKeeper, Oozie, Pig and Sqoop. In addition Apache Ambari delivers a dashboard for showing cluster health such as heatMaps and ability to view MapReduce, Pig and Hive applications visually along with features to diagnose their performance characteristics in a user-friendly manner.
- **Avro**: a data serialization system.
- **Cassandra**: climbable multi-master database with no single points of failure.
- **Chukwa**: for data gathering used to manage large distributed systems.

- **HBase:** scalable, distributed database that supports structured data storage for large tables.
- **Hive:** data warehouse arrangement that provides data summarization and ad hoc querying.
- **Mahout:** scalable machine learning and data mining archive.
- **Pig:** A top data-flow language and implementation framework for parallel computation.
- **Spark:** fast and general compute engine for Hadoop data. Spark provides a simple and expressive programming model that provides a extensive range of applications, including ETL, machine learning, stream processing, and graph computation.
- **Tez:** generalized data-flow programming framework, constructed on Hadoop YARN, which delivers a powerful and elastic engine to execute an arbitrary DAG of tasks to process data for both batch and communicating use-cases. Tez is being adopted by Hive, Pig and other frameworks in the Hadoop ecosystem, and also by other commercial software (e.g. ETL tools), to replace Hadoop MapReduce as the underlying execution engine.
- **ZooKeeper:** A high-performance coordination service for distributed applications. [10]

2.4.2 REAL-TIME DATA PROCESSING

Can be classified into two major ways:

- Solutions that try to reduce the overhead of MapReduce and make it faster to enable execution of jobs in less than seconds.
- Solutions that focus on providing a means for real-time queries over structured and unstructured big data using new optimized approaches.

2.4.2.1 IN-MEMOREY COMPUTING

There are two problems that cause slowness in Hadoop:

- The first problem in starting execution of jobs, is that it is not optimized for fast execution, scheduling, task assignment, code transfer to slaves. Beside job startup procedures are not designed and programmed to finish in less than seconds, because Hadoop is based on batch processing. To solve this problem job startup and task execution modules are redesigning.
- The second problem is in HDFS: it designed for high throughput data I/O rather than high performance I/O, and the HDFS is stored very large data blocks on hard disk drives and the Hadoop transfer rates between 100 and 200 megabytes per second (this mean even a simple search over the data will take minutes rather than seconds). And the *solution* to this problem is: In memory computing solution and it is used to solve distributing data among machines and reduce the time of reading data. This main memory has many features and some of this features is higher bandwidth, and Access latency is also much better. There are few in-memory computing solutions available like: Apache Spark GridGain, and XAP. And the Spark is both open source and free, but others are commercial.

“In-memory computing” does not mean the whole data should be kept in-memory, but means a distributed pool of memory. If this distributed pool of memory is available, then framework can use this memory for caching frequently used data. There for the whole job execution performance can be improved significantly.

Efficient caching can be effective when an iterative job is being executed. Both Spark and GridGain support this caching paradigm. Spark uses RDD and the RDD is primary abstraction means distributed collection of items. Spark can be easily combined with Hadoop and RDDs can be generated from data sources like HDFS and HBase. The in-memory computing feature of Spark enables it to compute data batches quicker than Hadoop.

In-memory caching can also help handling huge streaming data that can easily choke disk-based storages. [9]

2.4.2.2 REAL-TIME QUERIES OVER BIG DATA

Real time in big data is focused on interactivity rather than milliseconds response, but the real-time queries should respond in order of seconds and minutes rather than batch jobs which finish in hours and days.

2.4.3 STREAMING BIG DATA

Data streams now has many common examples such as, log streams, click streams, message streams, and event streams. However, the standard MapReduce like Hadoop framework is based on batch processing, and that means before the computation is started, all of the input data must be completely available on an input store, like, HDFS. After the framework process the input data, the output result is been available only when all of this computation is not done that means a MapReduce job execution is not continuous.

Today more application need to run continuously such as, query that detects some special anomalies from incoming events. This means today's applications need more streams. Unfortunately this stream processing is not available in MapReduce, but rather, this technique can partially handle streams

known as micro-batching. The idea is to give the stream as a form of sequence of small batch chunks of data. On small breaks, the incoming stream is full to a chunk of data and is delivered to batch system to be processed. There are some examples of MapReduce model especially realtime that support streaming processing like spark and GridGain.

In Spark the concept DStream support streaming which is represented as a sequence of RDDs. The architecture of stream processing in Spark is given in Figure (2.1).



Figure 2.1: illustrate processing in Spark

Spark technique is not support full streaming process, and there are a limited stream processing frameworks that are inherently designed for big data streams. The most two famous frameworks are Storm from Twitter, and S4 from Yahoo. Both frameworks using JVM and both process keyed streams. [9]

2.5 OTHER BIG DATA FRAMEWORKS

There are many open source data processing frameworks are being used today, in addition of Apache Hadoop and spark frameworks which we were implemented in this project we will talk about them and also we took randomly another two of Apache open source frameworks for processing big data and give short identification about them.

2.5.1 APACHE STORM

Apache Storm is” a free and open source distributed real time computation system”, Storm makes it easy to reliably process boundless streams of data, the same thing which Hadoop do for batch processing storm do for real time processing and it can be used with any programming language.

Storm can be used in many cases like: real time analytics, online machine learning, continuous computation, and more. It is fast: over a million tuples processed per second per node. Also storm is scalable, fault-tolerant, assurances your data will be processed, and is easy to set up and work. [11]

2.5.2 APACHE FLINK

“Is an open-source stream processing framework for distributed, high performing, always-available, and accurate data streaming applications”

Apache Flink has many features:

- 1- It is continuously processes datasets that are added to it constantly.
- 2- It runs on thousands of nodes with efficient and good latency.
- 3- The result which is delivered by Apache Flink is precise, even in the case of unordered or late arriving data.
- 4- It is fault-tolerant and “stateful” which it means can maintain a combination or summary of data that has been processed over time.
- 5- Can recover from failures while maintaining exactly-once application state [12].

2.6 SUMMARY

This chapter focused on Details about the Big Data science, some of big data processing types, and some examples of frameworks of big data.

Next chapter will show the previous studies and table illustrate that what studies agree with our research and what do not agree, and our opinion in each study.

CHAPTER THREE

LITERATURE REVIEW

3.1 INTRODUCTION

This chapter includes the previous studies, which related to this research, and table illustrate that what studies agree with this research and what do not agree, and our opinion in each study.

3.2 PREVIOUS STUDIES

3.2.1 FRAMEWORKS PERFORMANCE

3.2.1.1 ABOUT HDFS IN READ/WRITE

The traditional tools for processing and analyzing data, found it difficult to process and capture big data. Hadoop architecture consist of a file system called Hadoop Distributed File System (HDFS) which is an architecture used to store data. The paper focused on which kind of read and write way or technology would be selected and depending on what.

The paper talk about Hadoop and its two versions and concentrated on YARN, also talk about Apache Avro framework and its reliability on schemas. After that the paper explained the Sequence Files and its types, then it put the light on HBase and shows its major architectural components and the kind of its data store is a column-oriented, also the paper represents the HDFS and HBase properties.

As a result of paper, sequential file is used to deal with flat files and extract data form and put into them, and it is slower than normal file system. Also HDFS files regard as write-once and read-many files, and HBase is scale in terms of writes as well as total volume of data. Also it is good for structural data and the need of extraction in column manner rather than row by row. [13]

3.2.1.2 SPARK PERFORMANCE AND USABILITY

This paper talks about a group of people in Databricks who improve Spark performance and usability, so they deploy Spark to a wide range of organizations, they describe the challenges in Spark, also they show the needs of Spark users, and lastly they improve Spark based on users report.

Some of Spark challenges that they mentioned are: debugging and profiling, memory management ...etc.

To solve these challenges they describes three areas of works that tackle these challenges, these areas are: Engine Improvements, debugging tools.

They develop a more declarative API. This API is based on dataFrames. They also have outgoing work to improve Spark performance and usability. [14]

3.2.1.3 DATA PROCESSING USING HADOOP FRAMEWORK

According to this paper the author talk about big data and its characteristics and focuses on Hadoop framework which is used for capture, processing and analyzing big data .and also focus on two major components of Hadoop which are MapReduce and HDFS. Hadoop MapReduce framework is difficult to understand and it Needs time for execution. These problems solved by implemented Pig and Hive, so the author also talk about them. After that he discusses the modeling of Hadoop framework and its future work.

In conclusion the author presents the use of Hadoop in some domains. [16]

3.2.2 FRAMEWORKS PRPCESSING

3.2.2.1 BATCH AND STREAM PROCESSING ON SPARK AND FLINK FRAMEWORKS

The paper is focus in compare between batch API and stream API in both framework, Apache Spark and Flink. By performing repeated experiments for both and then extraction the result from the experiments.

Then the author reviewed related work and he suggested some recommendation to future work, he used Tera-sort benchmark tool for comparison and compare between the two frameworks in network usage and disk usage.

He found in network usage comparison that is Apache Flink have fixed rate in network data traffic and Apache Spark does not have this fixed rate in network traffic. In contrast when compared to the disk usage he found the attitude of the disk also reverses on attitude of the network , so Spark don't use the network at the beginning for reading data, so it caused late in the reading. Then used random bitstrings for streaming compared, and found that is the response time of Apache Flink is minimum than the response time of Apache Spark.

In conclusions he found that the Apache Flink framework is best in the streaming processing because it based on the concept of streaming, and Spark is better in the batch processing because it based on the concept of micro-batch processing. [15]

3.2.2.2 COMPARISON BETWEEN MICRO-BATCH AND STREAMING PROCESSING

The paper is about benchmarking between two open source platform for batch processing as well as streaming processing engine by using Amazon data ,also discuss these big data frameworks and show the limitations of Spark as the existing system and the advantages of Flink as the new system also talk about Spark streaming and Flink streaming .After that the author compare between Apache Spark and Apache Flink in many features and show the infrastructure statistics and data statistics which is used for comparison , the paper presents the performance of Flink streaming Vs Spark streaming, monthly distribution of reviews on Amazon Data, monthly average ratings of new Amazon reviews

In conclusion the author found that both Spark and Flink supply local connection with Hadoop and NoSQL Databases and able to process HDFS data. Also the author find Spark is slower than "Flink" but Spark is more famous and has strong community support and contributors, according to Amazon data he found Spark is 179.5% better than "Flink", and average time for processing With "Flink" is 240.3sec and Spark is 60.4sec. [17]

3.2.3 BENCHMARKING

3.2.3.1 COMPARING IN PERFORMANCE USING BENCHMARKS BETWEEN HADOOP, SPARK AND HAMAR:

According to this paper, author use benchmark to compare HADOOP, SPARK and HAMAR performances. He selected and ran PageRank, Word-Count, Sort, Tera-Sort, K-means and Naive Bayes

benchmarks on Hadoop and Spark runtime systems, and ran PageRank and Word-Count on HAMR runtime system.

And data generators provided in HiBench Benchmark suite. He measured the running time, maximum and average memory, CPU, usage and the throughput to compare the performances difference among these platforms for the six benchmarks.

As a result, the author found that Spark has brilliant performance on machine learning applications including K-means and Naive Bayes. For PageRank, Spark runs faster with small input size. Spark is faster on Word-Count. For Sort and Tera-Sort, Spark runs faster with large input. However, Spark consumes more memory capacity and the performance for Spark is restricted by the memory. HAMR is faster than Hadoop for both two benchmarks with improvements on CPU and memory usage. [7]

3.2.3.2 USING BENCHMARKING FOR STREAMING SYSTEMS:

By Looking at the paper in (Dec, 2015) the Yahoo Company made benchmark tool for comparing between the big data frameworks that based on streaming processing, that frameworks which represented is: Apache Storm - Apache Flink - Apache Spark.

Apache Storm and Flink, they similar that both of them based on the real-time streaming processing, but Apache Spark based on microbatch streaming processing.

Based on the comparison results made by Yahoo, it was found that the company used in the current status the Apache Storm framework, because the storm is very useful when it need the fast real-time system and high response time, conversely if you want a high throughput but you have delay in this case can use Apache Spark framework. [18]

▪ **THE FOLLOWING TABLE SHOWS OUR IMPRESSION OF THE PAPERS THAT ARE RELEVANT TO OUR RESEARCH**

Study name	Our Opinion
<input type="checkbox"/> About HDFS read and write.	<p>We agree with the author, the methodology of read and write depends on the dataset or the case which will process, and sequential files are slower and must be selected when the requirement required sequential processing only.</p>
<input type="checkbox"/> Spark performance and usability.	<p>They do great work to improve Spark performance, usability and API, and they worked on really challenges and important areas, these areas was problems that faces many developers who wanted to implement Spark to make use of their data and have less time to get their result, and achieve Spark power which concentrated in: in-memory processing and simple API.</p> <p>This paper is from 2015, and these improvements have been done on Spark 1.2, but now Spark 2 is available with more improvements and features.</p>

<p>□ Data processing using Hadoop Framework</p>	<p>We don't agree with author in his Opinion on Hadoop MapReduce, because it is not difficult to understand MapReduce job, and it takes more time in some cases.</p>
<p>□ Batch and stream processing on spark and Flink frameworks.</p>	<p>The author adopted in the comparison on data processing of the Spark in the disk, while instead of that he could have benefited from the feature of the Spark in-memory-processing.</p>
<p>□ Comparison between micro-Batch and streaming processing.</p>	<p>The results of the experiment is contradictory with the conclusion of the author because he said Flink is faster than Spark and the result shows the opposite.</p>
<p>□ Comparing in performance using benchmarks between Hadoop, Spark and Hamar.</p>	<p>This paper different from our project in two things:</p> <ul style="list-style-type: none"> • They compare HADOOP, SPRK and HAMAR, but we compare HADOOP and SPARK only. • They use HiBench Benchmark suite regarding workloads and we use BigBench Benchmark suite for generate structure data (E-commerce) and using it.

□ Using benchmarking for streaming systems.	We like to mention the uses of frameworks based on what is company need and the volume and kind of data.
---	--

Table 1: Table 3.1 illustrate our opinion.

3.3 SUMMARY

This chapter focused on related or previous studies in term of Big Data and Benchmark and processing. The next chapters will show the research methodology and tools and techniques used to develop the research's framework.

CHAPTER FOUR

TOOLS, TECHNIQUES

AND

RESEARCH METHODOLOGY

4.1 INTRODUCTION

This chapter consists of two sections, the first section concerns the tools and techniques used and the second section relates to the research methodology.

4.2 TOOLS AND TECHNIQUES

4.2.1 APACHE HADOOP

The Apache Hadoop software library is “a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models.” .Apache Hadoop ecosystem contain many component. This research focused on using two components from Apache Hadoop ecosystem, they are Hadoop Distributed File System (HDFS) and Hadoop MapReduce framework.

□ Hadoop Distributed File System (HDFS)

Hadoop Distributed File System is a distributed file system designed to run on commodity machines. HDFS is a high degree of fault-tolerant and it designed to be installed on low-cost hardware. Also it provides high data access to application data and is appropriate for applications that have big data sets. [19]

□ Hadoop MapReduce

- MapReduce is a programming paradigm, and it is the heart of Hadoop, it is a system for parallel processing on Big Data.
- The MapReduce has two basic functions: map and reduce. These two functions take the input and exit the output, the input is a set of key/value pairs, and the output is a list of key/value pairs (possibility

to be empty).The Execution of a MapReduce program involves two phases, first phase : each input pair is given to map function and a set of input pairs is produced with key and value, second phase : aggregated all the intermediate values that have the same key into a list, and then these list are given to a reduce function.

- To distribute MapReduce is implemented using architecture master/slave. The master machine role is assign tasks and controlling the slave machines. The master machine role is assignment of tasks and controlling the slave machines. The graphic in Figure (3.1) demonstrates the structure of MapReduce job: The input file is stored in a shared store (such as a distributed file system) and it split into chunks. First the implementation starts by giving copies of map and reduce functions code, then the master assigns all map and reduce tasks to workers. Any map worker reads the corresponding input split, and sends all of its pairs to map function and writes the results of the map function into intermediate files. After map phase is finished, the reducer workers read intermediate files, and send the intermediate pairs to reduce function, and finally write the pairs of result which has been produced by reduce tasks into final output files.

□ REASONS OF CHOOSING (HDFS) FOR STORING AND (MAPREDUCE) FOR PROCESSING

Hadoop is the most common and popular implementation of MapReduce.

Hadoop uses master/slave architecture that illustrate in Figure (4.1) by default, Hadoop stores input and output files on its distributed file system (HDFS). For example, it can also use NoSQL databases like HBase and Cassandra and even relational databases instead of HDFS. We have chosen HDFS because it is default file system of Hadoop, and our data is one big table which is not an entire database, and HDFS fits to our data. [9]

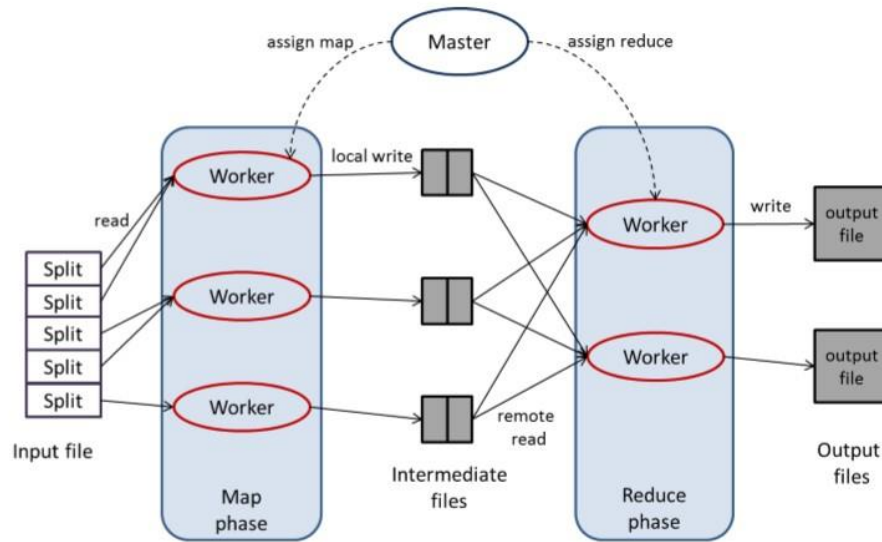


Figure 4.1: Architecture of MapReduce execution

4.2.2 APACHE SPARK

Apache Spark “is a fast and general engine for large-scale data processing “[20]. It provides applications that reuse set of data through parallel operations .In addition Spark support scalability and fault tolerance looks like Hadoop MapReduce [7].

Users take benefit of memory-centric computing structural design, which is provided in Spark. In addition extra intelligent optimization of user programs is enabled [21].

- **RESILIENT DISTRIBUTED DATASET(RDD)**

It is the fundamental concept and data structure of Spark, which is a group of fixed, distributed elements that can be worked in parallel [22].

When a job is assigned to Spark, the data is read from HDFS or other distributed file systems, and it is then cached in memory .RDD helps to reduce

reading from and writing to disk which participates in speeding up the processing, RDD could not update so any changes leads to create a new partition. Apache Spark runs APIs in Java, Scala, Python and R shells.

- **RUNNING THREE CLUSTER MANAGER USING SPARK**

- Standalone mode: in easy way can be used in single node, in this mode the cluster can lunched by hand or by launch script, this mode supported by Spark distribution

- Mesos mode: Mesos: is delivered by Apache designed to manage the clusters

- YARN mode: requires Hadoop2.0 or any Hadoop version after Hadoop2.0. [7]

- **APACHE SPARK AS A BEST CHOISE**

It is first a competitor to Hadoop MapReduce and it solved latency in Hadoop MapReduce, and it's approximate that always against it.

4.2.3 APACHE HIVE

It is a software for data warehouse that helps reading, writing, and managing large datasets, that located in a distributed storage. It is build to work on top of Apache Hadoop.

Apache Hive provides tools to allow easy access to data via SQL, so as to support data warehousing jobs such as extract/transform/load (ETL), and reporting. It enable enable access to files stored in Apache HDFS, or another data storage system like HBase.

Apache Hive execute queries by Hadoop MapReduce, so as allows user to get used with SQL by HiveQL in data query. Also programmers who are familiar with Hadoop MapReduce framework can be able to connect their own mappers and reducer to execute more advanced analysis by HiveQL that may not be provided by natural skills of language [23].

□ THE CHOISE OF APACHE HIVE

Hive provides an SQL like interface called HiveOL, which makes your work and even query easier. With this interface you can create tables in Hive and store data in it, and even run an operations on tables created [24].

In our research we used Apache Hive to execute queries over data that has been processed by Hadoop MapReduce and Spark.

4.2.4 BIGDATABENCH

BigDataBench offers several (parallel) big data generation tools-BDGS- to generate big data, from small-scale real-world data while preserving their original characteristics. For example, on an 8-node cluster system, BDGS generates 10 TB data in 5 hours. For the same workloads [25].

THE PROPERITES OF BIGDATABENCH SUITE

In our research we used BigDataBench because it has many properties:

- There is a need to be specific when use Bigdatabench.
- It had eight real scalable data sets that are extracted from real-world data sets.
- For the same workload specification, diverse implementations using competitive techniques are provided, such as Hadoop and Spark
- It has six workload types, include: Streaming, Offline Analytics.
- There are 42 workloads in the specification in Bigdatabench.

Bigdatabench properties that we mentioned are increasing by updating the versions of Bigdatabench [26].

In our research we used BDGS to generate Amazon data, as structural nature, and search engine data as an example for unstructured data.

4.3 RESEARCH METHODOLOGY

Here you will find the steps and activities that we have done to reach our objectives:

- Survey.
- Prepare the environment.
- Configuring Hadoop and Spark in all machines.
- Formatting name-node in master node.
- Starting Hadoop and Spark and hive Services.
- Data generation.
- Running jobs.

4.3.1 SURVEY

4.3.1.1 SURVEY PURPOSES

- To prove there is big data in Sudan.
- To investigate whether there is big data and how its existence could make a real problem in companies.
- To find out whether big data solutions have been used in Sudan.

4.3.1.2 SAMPLE DISTRIBUTION AND FILLING UP

We divided Sudan into three sectors: services sector, telecommunication sector and banks sector. We have chosen these sectors because companies belonging to these sectors may have big data. We took a random sample of companies from these sectors. Then we design the questionnaire by google form. And ask the employee of this companies to help us getting information needed. Finally the questionnaire has been filled up and discussed with those employees.

4.3.1.3 SURVEY QUESTIONS

As shown in appendix (B), the questions were very clear and most of them cannot accept multiple answers.

We ask about latency in question 5 to see if the companies suffer from its big data, for example in decision making or reports generating. We also asked about batch processing and parallel processing, to see the most one used in Sudan if existing. There are two questions that aim to know the mechanism of storage in companies. There are also some questions that aim to know which framework is used to process this big data, if it exist.

4.3.1.4 SURVEY RESULTS

These are the questions and the result of each one:

company name :

7 responses

وزارة التعليم العالي
الإيصال الإلكتروني
سوداني
MTN
بنك الخرطوم
Central Bank of Sudan
Zain Sudan

Figure 4.2: illustrate Question 1

Employee name :

7 responses

بابكر حسين
محمد عبد الرحيم
محمد المجتبي (developer)
علي طارق
أحمد مختار
Abubakar Ahmed Elhussien
Elrasheed Elnoman Mohamed Abdalla

Figure 4.3: illustrate Question 2

Department name :

7 responses

مدير قسم النظم التطبيقية
BI فريق ال
BI
manager bi
مدير وحدة البنية التحتية و العمليات
Information Technology
Business Intelligence and Data Warehouse

Figure 4.4: illustrate Question 3

Did your company has big data(e.g Billion record or 1 Tera volume...etc)

7 responses

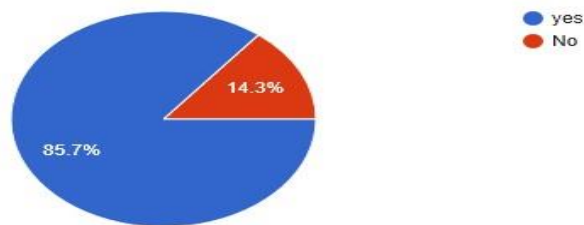


Figure 4.5: illustrate Question 4

Did this big data make latency in company's decisions(e.g delaying reports or delaying decisions ...etc)?

7 responses

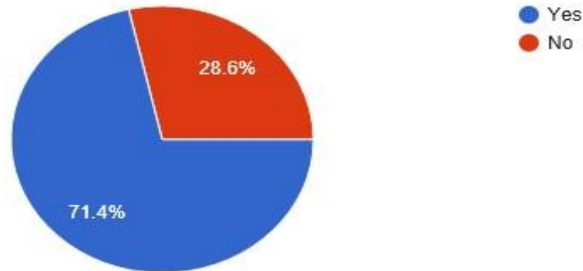


Figure 4.6: illustrate Question 5

Which techniques do you use to process this data?

5 responses

RDBMS
Relational Database
RDBMS , DataWarehouse
oracle RDB , Tuning
Oracle BI and DWH suits

Figure 4.7: illustrate Question 6

did you use Batch processing ?

7 responses

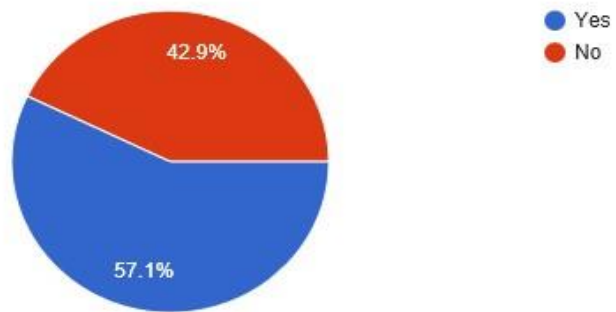


Figure 4.8: illustrate Question 7

did you use parallel processing ?

7 responses

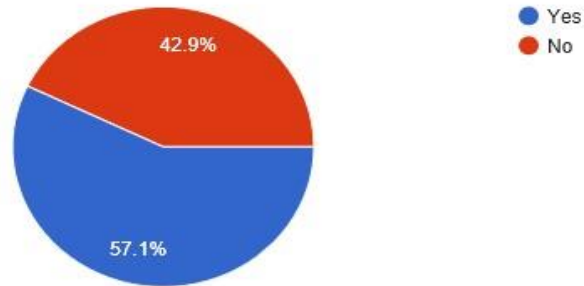


Figure 4.9: illustrate Question 8

where you store this big data(distributed or central)?

7 responses

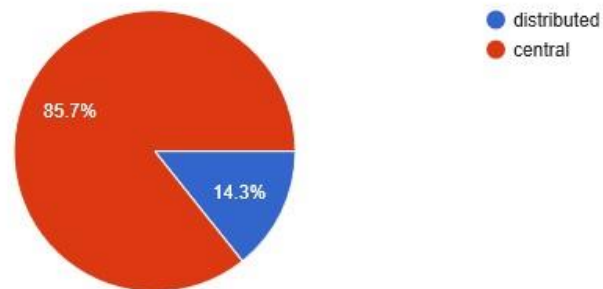


Figure 4.10: illustrate Question 9

Did you store data in HDFS?

7 responses

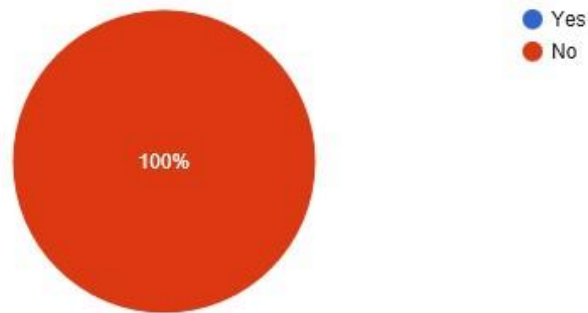


Figure4.11: illustrate Question 10

Did you use Hadoop MapReduce to implement this big data?

7 responses

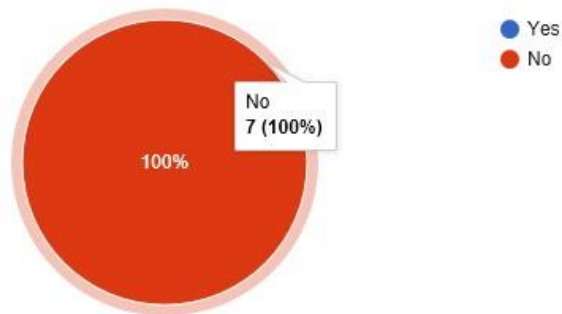


Figure 4.12: illustrate Question 11

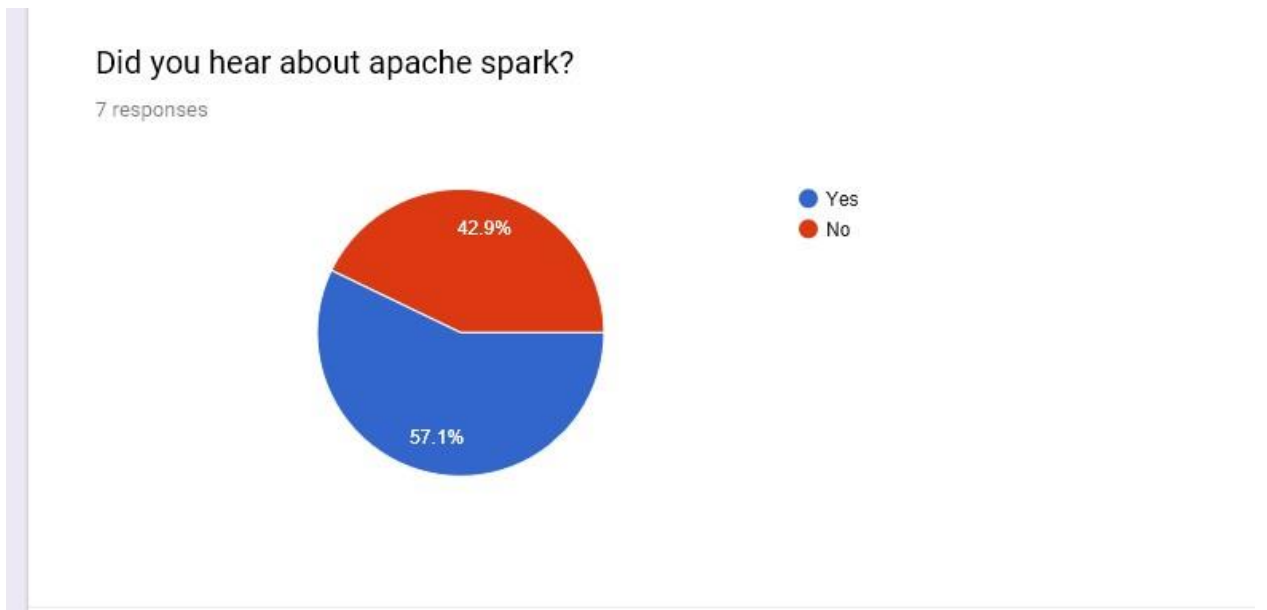


Figure 4.13: illustrate Question 12

Did you use apache spark framework to implement this big data?

7 responses

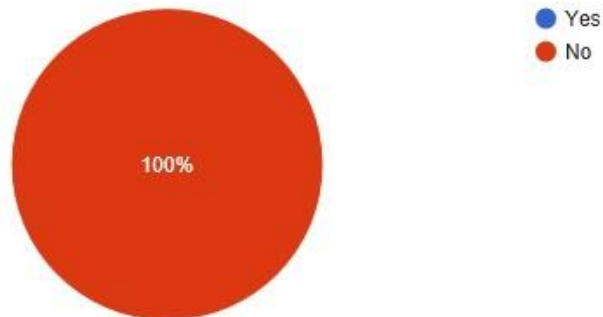


Figure 4.14: illustrate Question 13

HINT:

According to these survey results and interviews that we done, we discovered that a big data is really a big problem in some companies, especially in telecom companies, and found that these companies are starting to address big data concept this year, but they suffer from decrease of knowledge and team practice and expertness, based on this results, we do our research to learn and implement two big data solutions framework (Hadoop MapReduce and Spark) to help these companies to implement big data solutions and make use of their big data.

4.3.2 PREPARE THE ENVIROMENT

In this research the work was configured in 2 environments:

- **Multi-node cluster in a distributed environment consist of 3 machines(nodes):**
 - a. Master node (Containing Name-Node and Data-Node):
 - Processor: dual core, 2.10GHz, 2.10 GHz.
 - Installed memory (RAM): 4.00 GB.
 - Disk: 298.09 GB.
 - b. slave node (Containing DataNode):
 - Processor: core i3, 1.80GHz, 1.80 GHz.
 - Installed memory (RAM): 4.00 GB.
 - Disk: 465.75 GB.
 - c. slave node (Containing DataNode):
 - Processor: dual core, 2.20GHz, 2.20 GHz.
 - Installed memory (RAM): 4.00 GB.
 - Disk: 465.75 GB.

- **Multi-node cluster in a distributed environment consist of 2 machines(nodes):**
 - a. Master node (Containing Name-Node and Data-Node):
 - Processor: core i3, 1.80GHz, 1.80 GHz.
 - Installed memory (RAM): 4.00 GB.
 - Disk: 465.75 GB.
 - b. Slave node (containing DataNode):
 - Processor: dual core, 2.10GHz, 2.10 GHz.
 - Installed memory (RAM): 4.00 GB.
 - Disk: 298.09 GB.

4.3.3 CONFIGUREING MACHINES CONNECTION

In cluster mode implementation we connect machines with Ethernet network. Firstly, we mapped nodes by editing (hosts) file in “etc” folder in all machines, by specifying IP address for each machine followed by hostname of the machine.

Secondly configure SSH login in each node, so that each machine can enter to another one without password, by commands shown in figure (4.15).

```
$ sudo apt-get install openssh-server
$ ssh-keygen -t rsa
$ ssh-copy-id -i ~/.ssh/id_rsa.pub hdusr@master
$ ssh-copy-id -i ~/.ssh/id_rsa.pub hdusr@slave1
$ ssh-copy-id -i ~/.ssh/id_rsa.pub hdusr@slave2
$ chmod 0600 ~/.ssh/authorized_keys
$ sudo reboot
```

Figure 4.15: SSH steps

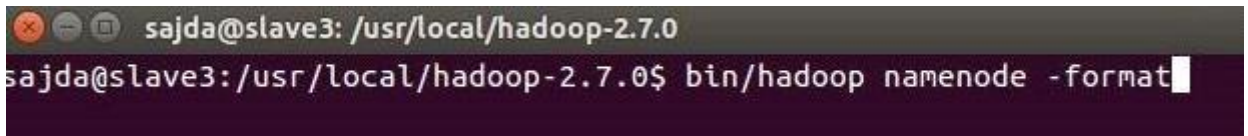
Configuring SSH in each node must be done before configuring Hadoop and Spark frameworks, because master node must have access to all slave nodes, so it can distribute data and work in slave nodes.

4.3.3.1 CONFIGURING “HADOOP”, “SPARK” AND “HIVE”

We wrote documents and share videos on YouTube for the right and easiest way to configure “Hadoop” and “Spark”, and the preinstalled programs needed by “Hadoop” and “Spark”. These documents and videos are found in appendix (A).

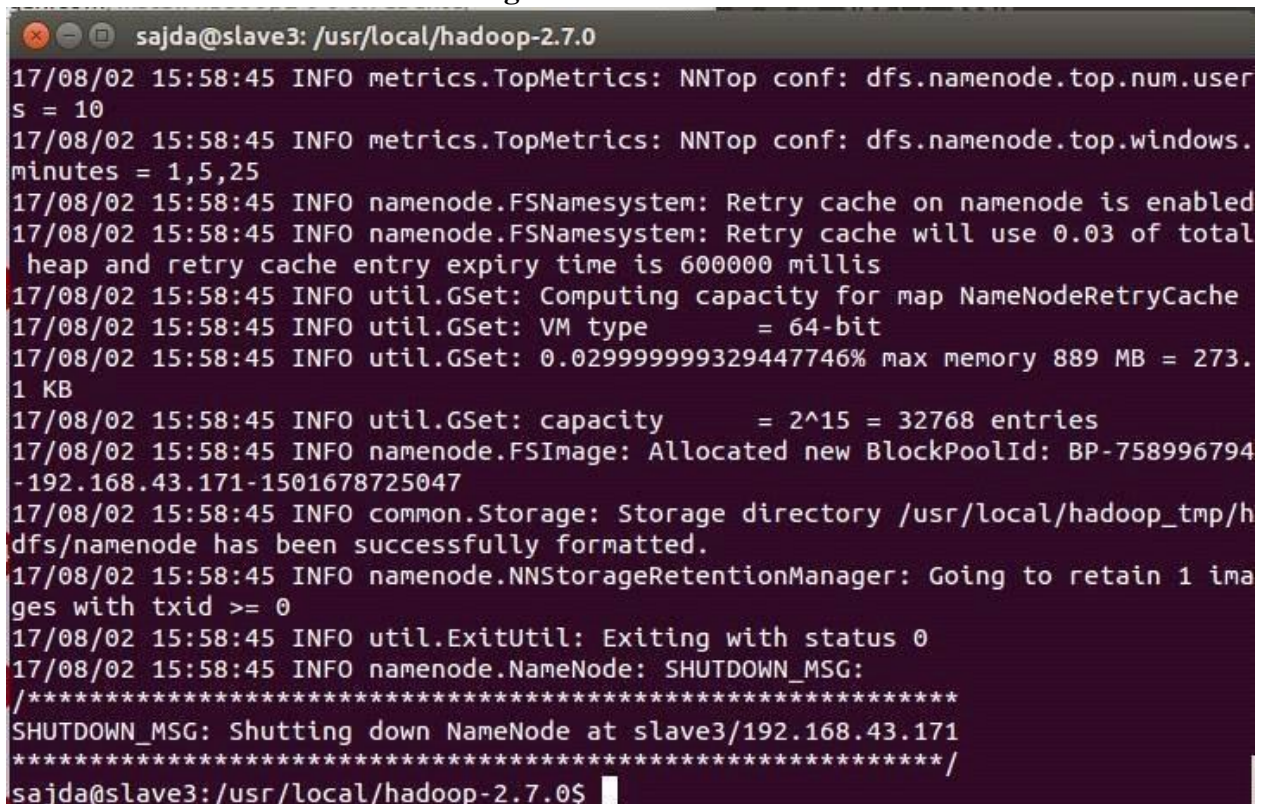
- **FORMATING NAMENODE IN MASTER NODE :**

It is an important step when configuring Hadoop. It empties Hadoop file system from any data. We formatted it using command as shown in figure (4.16), and the feedback after command is shown on figure (4.17)



```
sajda@slave3: /usr/local/hadoop-2.7.0$ bin/hadoop namenode -format
```

Figure 4.16: format NameNode



```
17/08/02 15:58:45 INFO metrics.TopMetrics: NNTop conf: dfs.namenode.top.num.users = 10
17/08/02 15:58:45 INFO metrics.TopMetrics: NNTop conf: dfs.namenode.top.windows.minutes = 1,5,25
17/08/02 15:58:45 INFO namenode.FSNamesystem: Retry cache on namenode is enabled
17/08/02 15:58:45 INFO namenode.FSNamesystem: Retry cache will use 0.03 of total heap and retry cache entry expiry time is 600000 millis
17/08/02 15:58:45 INFO util.GSet: Computing capacity for map NameNodeRetryCache
17/08/02 15:58:45 INFO util.GSet: VM type = 64-bit
17/08/02 15:58:45 INFO util.GSet: 0.029999999329447746% max memory 889 MB = 273.1 KB
17/08/02 15:58:45 INFO util.GSet: capacity = 2^15 = 32768 entries
17/08/02 15:58:45 INFO namenode.FSImage: Allocated new BlockPoolId: BP-758996794-192.168.43.171-1501678725047
17/08/02 15:58:45 INFO common.Storage: Storage directory /usr/local/hadoop_tmp/hdfs/namenode has been successfully formatted.
17/08/02 15:58:45 INFO namenode.NNStorageRetentionManager: Going to retain 1 images with txid >= 0
17/08/02 15:58:45 INFO util.ExitUtil: Exiting with status 0
17/08/02 15:58:45 INFO namenode.NameNode: SHUTDOWN_MSG:
/*****
SHUTDOWN_MSG: Shutting down NameNode at slave3/192.168.43.171
*****/
sajda@slave3: /usr/local/hadoop-2.7.0$
```

Figure 4.17: Successfully formatted

- **STARTING HADOOP AND SPARK AND HIVE SERVICES**

Master node is responsible for starting Hadoop and Spark services (see chapter two).

We cannot do any job in “Hadoop” or “Spark” without starting these services. Figure (4.18) showing starting these services.

```

slave1: starting namenode, logging to /usr/local/hadoop-2.7.0/logs/hadoop-hdusr-
namenode-slave1.out
slave1: starting datanode, logging to /usr/local/hadoop-2.7.0/logs/hadoop-hdusr-
datanode-slave1.out
slave3: starting datanode, logging to /usr/local/hadoop-2.7.0/logs/hadoop-hdusr-
datanode-slave3.out
slave22: starting datanode, logging to /usr/local/hadoop-2.7.0/logs/hadoop-hdusr-
datanode-slave22.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /usr/local/hadoop-2.7.0/logs/had
oop-hdusr-secondarynamenode-slave1.out
17/10/17 14:34:33 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
ry for your platform... using builtin-java classes where applicable
starting yarn daemons
starting resourcemanager, logging to /usr/local/hadoop-2.7.0/logs/yarn-hdusr-res
ourcemanager-slave1.out
slave1: starting nodemanager, logging to /usr/local/hadoop-2.7.0/logs/yarn-hdusr-
nodemanager-slave1.out
slave3: starting nodemanager, logging to /usr/local/hadoop-2.7.0/logs/yarn-hdusr-
nodemanager-slave3.out
slave22: starting nodemanager, logging to /usr/local/hadoop-2.7.0/logs/yarn-hdusr-
nodemanager-slave22.out
hdusr@slave1: /usr/local/hadoop-2.7.0$

```

Figure 4.18: Starting Hadoop daemon

4.3.3.2 DATA GENERATION

We try to use data set from real case in Sudan, but we did not succeed for the following reasons:

- Big data trend is novel in Sudan, and just big companies started thinking of it. Unfortunately no company has a complete implementation of real big data solution for its big data.
- In Sudan big companies are afraid of giving students its data.
- We have requested big data set from (X) company, at the beginning they have accepted our request, but later they decided to reject our request because of data sensitivity.

In this research we have generated E-commerce data as structural data, and search engine data as unstructured data.

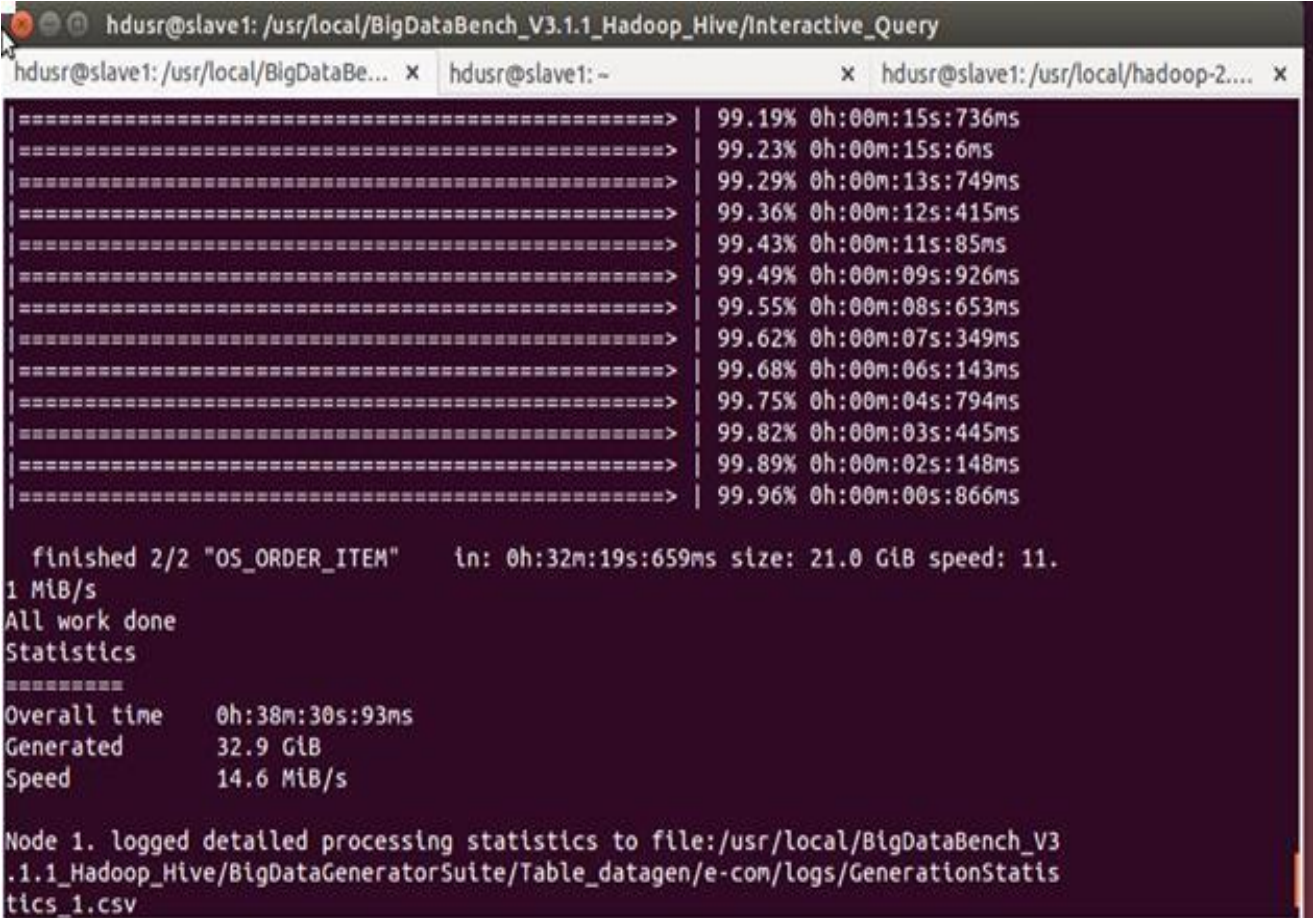
- **DATA DESCRIPTION**

1. Search engine data set

- Text data has been generated, which is Wikipedia Entries.
- Its one-gigabyte data.

2. E-commerce data set

- Table data has been generated, which is E-commerce Transaction Data.
- Its 30-gigabyte data as shown in figure(4.19)



```
hdusr@slave1: /usr/local/BigDataBench_V3.1.1_Hadoop_Hive/Interactive_Query
hdusr@slave1: /usr/local/BigDataBe... x hdusr@slave1: ~ x hdusr@slave1: /usr/local/hadoop-2.... x

=====> | 99.19% 0h:00m:15s:736ms
=====> | 99.23% 0h:00m:15s:6ms
=====> | 99.29% 0h:00m:13s:749ms
=====> | 99.36% 0h:00m:12s:415ms
=====> | 99.43% 0h:00m:11s:85ms
=====> | 99.49% 0h:00m:09s:926ms
=====> | 99.55% 0h:00m:08s:653ms
=====> | 99.62% 0h:00m:07s:349ms
=====> | 99.68% 0h:00m:06s:143ms
=====> | 99.75% 0h:00m:04s:794ms
=====> | 99.82% 0h:00m:03s:445ms
=====> | 99.89% 0h:00m:02s:148ms
=====> | 99.96% 0h:00m:00s:866ms

finished 2/2 "OS_ORDER_ITEM"   in: 0h:32m:19s:659ms size: 21.0 GiB speed: 11.
1 MiB/s
All work done
Statistics
=====
Overall time   0h:38m:30s:93ms
Generated     32.9 GiB
Speed         14.6 MiB/s

Node 1. logged detailed processing statistics to file:/usr/local/BigDataBench_V3
.1.1_Hadoop_Hive/BigDataGeneratorSuite/Table_datagen/e-com/logs/GenerationStatis
tics_1.csv
```

Figure 4.19: generated 30-gigabyte data

- This data is divided into two text files, ORDERS.txt and ORDERS_ITEMS.txt.

- o ORDERS.txt file: consist of 3 attributes, order_id (integer), buyer_id (integer) and create_date (string). This file has 420,000,000 record.
- o Figure (4.20) shows the format of the file ORDERS_ITEMS.txt as it downloaded.

```

0|0|2007-04-24|
1|1|2011-05-06
2|2|2011-06-15
3|3|2010-08-16
4|4|2007-09-02
5|5|2011-10-10
6|6|2010-08-05
7|7|2008-05-31
8|8|2008-07-21
9|9|2008-07-31
10|10|2009-02-09
11|11|2009-08-21
12|12|2007-09-25
13|13|2010-09-01
14|14|2010-08-30
15|15|2011-09-16
16|16|2010-08-28
17|17|2007-05-05
18|18|2011-12-08
19|19|2010-11-28
20|20|2011-04-17
21|21|2011-01-16
22|22|2011-10-09
23|23|2010-10-07
24|24|2011-11-07
25|25|2007-04-19
26|26|2008-05-30
27|27|2008-06-22
28|28|2011-06-23
29|29|2011-07-01

```

Figure 4.20: ORDERS.txt file

- o ORDERS_ITEMS.txt file: consist of 5 attributes, item_id (integer), order_id (integer), goods_id (integer), goods_number (double), goods_price (double) and goods_amount (double). This file has 420,000,000 record also.
- o Figure (4.21) shows the format of the file ORDERS_ITEMS.txt as it downloaded.


```

0|4210351682594|0|567|661.40|375117.12
1|4308856748175|1|601|275.27|165442.89
2|4407361813756|2|843|338.43|285618.31
3|4505866879337|3|423|923.01|390460.66
4|4604371944918|4|424|862.29|366151.53
5|4702877010499|5|99|28.64|2850.46
6|4801382076080|6|561|544.24|305651.01
7|4899887141661|7|972|717.62|698228.44
8|4998392207242|8|100|252.17|25407.78
9|96897272823|9|712|618.97|440931.92
10|195402338404|10|4|921.31|4277.48
11|293907403985|11|666|347.65|231790.68
12|392412469566|12|625|649.86|406562.44
13|490917535147|13|390|601.99|235281.48
14|589422600728|14|410|441.31|181267.49
15|687927666309|15|282|17.84|5041.28
16|786432731890|16|626|899.59|563428.22
17|884937797471|17|947|173.99|164909.55
18|983442863052|18|724|120.60|87360.19
19|1081947928633|19|799|494.25|395238.07
20|1180452994214|20|970|204.08|198085.33
21|1278958059795|21|532|726.45|386942.86
22|1377463125376|22|340|878.88|299668.71
23|1475968190957|23|2|759.89|1720.07
24|1574473256538|24|370|924.65|342427.39
25|1672978322119|25|578|959.32|555284.82

```

Figure 4.21: ORDERS_ITEM.txt file

4.3.4 RUNNING JOBS

Two jobs have been run on that data generated, these jobs are: Word-Count on Hadoop cluster and select query on Hadoop and Spark. By these jobs, data processing concept has been implemented on Hadoop and Spark.

4.3.4.1 Running Word-Count Workload on Hadoop cluster

Word-Count is a workload from Microbenchmarks suite, its function is to count the appearance of each word in the input file, and it work on text file as an input. This workload is a script file as shown in the figure (4.22).

```

run_MicroBenchmarks.sh x
WORK_DIR='pwd'
echo "WORK_DIR=$WORK_DIR data should be put in $WORK_DIR/data-MicroBenchmarks/in"

algorithm=( sort grep wordcount)
if [ -n "$1" ]; then
  choice=$1
else
  echo "Please select a number to choose the corresponding Workload algorithm"
  echo "1. ${algorithm[0]} Workload"
  echo "2. ${algorithm[1]} Workload"
  echo "3. ${algorithm[2]} Workload"
  read -p "Enter your choice : " choice
fi

echo "ok. You chose $choice and we'll use ${algorithm[$choice-1]} Workload"
Workloadtype=${algorithm[$choice-1]}

if [ "$Workloadtype" == "xsort" ]; then
  ${HADOOP_HOME}/bin/hadoop fs -rmr ${WORK_DIR}/data-MicroBenchmarks/out/sort
  time ${HADOOP_HOME}/bin/hadoop jar ${HADOOP_HOME}/share/hadoop/mapreduce/hadoop-mapreduce-examples-*.jar sort /sort-data /sort-data/output
elif [ "$Workloadtype" == "xgrep" ]; then
  ${HADOOP_HOME}/bin/hadoop fs -rmr ${WORK_DIR}/data-MicroBenchmarks/out/grep
  time ${HADOOP_HOME}/bin/hadoop jar ${HADOOP_HOME}/share/hadoop/mapreduce/hadoop-mapreduce-examples-*.jar grep ${WORK_DIR}/data-MicroBenchmarks/in ${WORK_DIR}/data-MicroBenchmarks/out/grep a*xyz
elif [ "$Workloadtype" == "xwordcount" ]; then
  ${HADOOP_HOME}/bin/hadoop fs -rmr ${WORK_DIR}/data-MicroBenchmarks2/out/wordcount
  time ${HADOOP_HOME}/bin/hadoop jar ${HADOOP_HOME}/share/hadoop/mapreduce/hadoop-mapreduce-examples-*.jar wordcount ${WORK_DIR}/data-MicroBenchmarks2/in ${WORK_DIR}/data-MicroBenchmarks2/out/wordcount

  echo "unknown cluster type: $clustertype"
fi

```

Figure 4.22: Run_Microbenchmarks.sh file

WordCount Functionality Description

To implement the wordCount Functionality, The first step NameNode stores the input in HDFS on the background. Secondly word-count job is performed, to operate map reduce job on Hadoop cluster. The last step is to collect result and store it in a file in HDFS.

Machines performance data were collected while the word-count job is running. We choose running time, physical memory consumption and CPU time spent as operators, because they are the standard measurements to show the performance of each job. A performance data are collected using JobHistory user interface, which is a Hadoop service used to track MapReduce jobs. The first JobHistory service in Hadoop is started as shown in figure (4.23), and then open it in the browser. It is started in port 19888 as shown in figure (4.24).

```

hdusr@slave1: /usr/local/hadoop-2.7.0/sbin
hdusr@slave1:~$ cd /usr/local/hadoop-2.7.0/sbin
hdusr@slave1:/usr/local/hadoop-2.7.0/sbin$ ./mr-jobhistory-daemon.sh start historyserver
starting historyserver, logging to /usr/local/hadoop-2.7.0/logs/mapred-hdusr-historyserver-slave1.out
hdusr@slave1:/usr/local/hadoop-2.7.0/sbin$

```

Figure 4.23: start JobHistory service



Figure 4.24: JobHistory UI

4.3.4.2 Run “select” query on E-Commerce data set (described above) using Hadoop MapReduce and Spark

This “select” job was implemented on the same cluster, when using the two frameworks.

□ “Select” Query in Hadoop

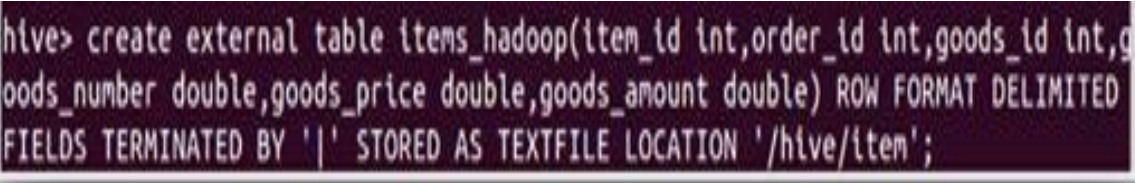
The first step was to store the input files in “HDFS” using commands as shown in figure (4.25). Figure (4.25) show that, firstly create a folder in HDFS called Hive and create 2 folders inside it, item and order

folder, then bring the text files from path where it stored through generation step, and put the ORDERS.txt file in order file on the HDFS, and the ORDERS_ITEMS.txt file in items file on the HDFS.

```
cd $BigdataBench_Home/Interactive_Query
hadoop dfs -rmr /hive
hadoop dfs -mkdir -p /hive/item
hadoop dfs -mkdir -p /hive/order
hadoop dfs -put $BigdataBench_Home/Interactive_Query/data2/OS_ORDER.txt /hive/order/
hadoop dfs -put $BigdataBench_Home/Interactive_Query/data2/OS_ORDER_ITEM.txt /hive/item/
```

Figure 4.25: create a folder in HDFS called Hive

Then by using Hive we have created a table called items_hadoop consists of 6 columns (item_id, order_id, goods_id, goods_number, goods_price and goods_amount), with hiveQL command as shown in figure (4.26).



```
hive> create external table items_hadoop(item_id int,order_id int,goods_id int,goods_number double,goods_price double,goods_amount double) ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' STORED AS TEXTFILE LOCATION '/hive/item';
```

Figure 4.26: created table in hive

The next step was to do the select Query over “items_hadoop” table, using hiveOL command as shown in figure (4.27).

```
hive> create table items_goods as select goods_price from items_hadoop;
```

Figure 4.27: Select Query in hive

Then MapReduce started doing the select job, as shown in figure (4.28).

```
hive> create table items_goods as select goods_price from items_hadoop;
FAILED: SemanticException 0:0 Error creating temporary folder on: hdfs://slave1:
9000/user/hive/warehouse. Error encountered near token 'TOK_TMP_FILE'
hive> create table items_goods as select goods_price from items_hadoop;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the futu
re versions. Consider using a different execution engine (i.e. tez, spark) or us
ing Hive 1.X releases.
Query ID = hdusr_20171016193439_161ec424-45e2-4411-b1aa-820dca8a27cf
Total jobs = 3
Launching Job 1 out of 3
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1508170046109_0001, Tracking URL = http://slave1:8088/proxy/a
pplication_1508170046109_0001/
```

Figure 4.28: Hadoop MapReduce started doing the select job

Finally the result was fetched, and the job information was saved in “jobHistory” interface, you will see these results in chapter 5.

□ Select query in Spark

Firstly starting “Spark-shell” application, as shown in figure (4.29) which is an application in Spark framework that enables you to write an application to be performed by Spark using Scala, java, or R languages.

On this research we used Scala to write a program in Spark to do the select Query.

Secondly we created “SparkContext” to initialize “hiveContext” in Spark shell, which enable to write queries and interact with hive “metastore” using “HiveQL” shown in figure (4.30)

Then used the Spark context value to create table called “items_Spark” with attributes: (item_id (integer), order_id (integer), goods_id (integer), goods_number (double), goods_price (double) and goods_amount (double)), as shown in figure (4.31), the system automatically create a folder called “Sparkwarehouse” to store tables on

it. Then we load the data stored in “ORDERS_ITEM.txt” file as shown in figure (4.32). After that we wrote select query on table “items_Spark” and stored the result on “resu” variable, shown in figure (4.33).

Finally showing the result using “show ()” method, and then went to “SparkJobs” interface and tracked this job information, it is started when we start Spark services, in port “4040”.

```
spark session available as 'spark' .
Welcome to

  ____
 /  __ \
| |  | |
| |__| |
|  __ /
| |  | |
|_|  |_|
version 2.1.1

Using Scala version 2.11.8 (OpenJDK Server VM, Java 1.7.0_151)
Type in expressions to have them evaluated.
Type :help for more information.

scala> val sqlContext = new org.apache.spark.sql.hive.HiveContext(sc)
warning: there was one deprecation warning; re-run with -deprecation for details
sqlContext: org.apache.spark.sql.hive.HiveContext = org.apache.spark.sql.hive.Hi
veContext@4da485
```

Figure 4.29: Start Spark shell

```
scala> val sqlContext = new org.apache.spark.sql.hive.HiveContext(sc)
warning: there was one deprecation warning; re-run with -deprecation for details
sqlContext: org.apache.spark.sql.hive.HiveContext = org.apache.spark.sql.hive.Hi
veContext@4da485

scala> █
```

Figure 4.30: write queries and interact with hive

```
scala> sqlContext.sql("CREATE TABLE IF NOT EXISTS items_spark(item_id int,order_
id int,goods_id int,goods_number double,goods_price double,goods_amount double)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' LINES TERMINATED BY '\n'")
res0: org.apache.spark.sql.DataFrame = []

scala>
```

Figure 4.31: Spark context value

```
scala> sqlContext.sql("LOAD DATA LOCAL INPATH '/usr/local/BigDataBench_V3.1.1_Ha
doop_Hive/Interactive_Query/data2/OS_ORDER_ITEM.txt' INTO TABLE items_spark")
res1: org.apache.spark.sql.DataFrame = []
scala>
```

Figure 4.32: load the data stored in “ORDERS_ITEM.txt”

```
scala> val resu = sqlContext.sql("FROM items_spark SELECT goods_price")
resu: org.apache.spark.sql.DataFrame = [goods_price: double]
scala>
```

Figure 4.33: store the result on “resu” variable

After this “select” job, we searched a lot for tool that measure machines performance while the job is running on the background, we tried “collect and Graphana” tool, which is used for monitoring machine resources (processor, RAM, Disk) performance, but we found that it monitor a single machine, and we are doing our job on a “cluster” consist of 3 machines. Then we tried “htop” service on Ubuntu, which read the measurements of machine while “processes” is running, but we discovered that when map Reduce or Spark run a job; this job started multiple processes in Ubuntu, it executed many java script files, for that reason we could not track jobs using this service (htop) because “htop”, giving results to single running process. For these reasons we used Hadoop service “JobHistory” to track MapReduce job, and “SparkJobs” interface to track Spark job.

4.4 SUMMARY

This chapter shows the main tools and techniques that have been used to achieve the goals of this project, and explained the research methodology that used to achieve goals. In next chapter we will show you the results has been collected from two jobs (Word-Count and select Query), and the comparison results.

CHAPTER FIVE

RESULTS

AND

RECOMMENDATIONS

5.1 INTRODUCTION

Last chapter describes the work that have been done and the tools and techniques used, it describes the two jobs (“Word-Count” and “Select” query) that have been done on the cluster using Hadoop MapReduce and Spark. This chapter will show you results that have collected when applying the two Jobs to Hadoop MapReduce and Spark, then the comparison results between the two frameworks. And also shows the recommendations.

5.2 RESULTS

These are the results that have collected when running jobs on Hadoop MapReduce and Spark.

5.2.1 PERFORMANCE RESULTS OF

“WORD-COUNT” JOB ON ‘Hadoop MapReduce’ CLUSTER

After running (Word-Count) on Hadoop cluster using Hadoop MapReduce framework, the result is stored in output file in the (HDFS), the output was in format of lines, and each line shows the word and its appearance in the text input file. Then get the resources measurements from “JobHistory” interface, we have chosen these three measurements to be shown in this chapter, but actually “JobHistory” interface gives many other measurements.

The result of “Word-Count” job take these consumptions while running: (Running time = 4 mins, 22 sec, Physical memory (bytes) = 2738606080, CPU time spent (ms) = 580,700)

5.2.2 PERFORMANCE RESULTS OF “Select” QUERY USING HADOOP MAPREDUCE

Resources measurements have get from “JobHistory” interface while the “select” job is running as shown in figure (5.1).

		Name	Map	Reduce	Total
Job Counters		Data-local map tasks	0	0	3
		Killed map tasks	0	0	1
		Launched map tasks	0	0	4
		Other local map tasks	0	0	1
		Total megabyte-seconds taken by all map tasks	0	0	297,588,736
		Total time spent by all map tasks (ms)	0	0	290,614
		Total time spent by all maps in occupied slots (ms)	0	0	290,614
		Total vcore-seconds taken by all map tasks	0	0	290,614
	Map-Reduce Framework		CPU time spent (ms)	51,240	0
		Failed Shuffles	0	0	0
		GC time elapsed (ms)	1,199	0	1,199
		Input split bytes	789	0	789
		Map input records	14,000,000	0	14,000,000
		Map output records	0	0	0
		Merged Map outputs	0	0	0
		Physical memory (bytes) snapshot	456,957,952	0	456,957,952
		Spilled Records	0	0	0
		Total committed heap usage (bytes)	232,521,728	0	232,521,728
		Virtual memory (bytes) snapshot	1,495,048,192	0	1,495,048,192
		Name	Map	Reduce	Total

Figure 5.1: measurements was saved in “JobHistory”

➤ RESULTS SNAPSHOT OF “select” QUERY

Running time: 1mins, 42sec

CPU time spent (ms):

- Map job takes = 51,240
- Reduce job takes = 0
- AVG= 51,240

Physical memory (bytes):

- Map job used = 456,957,952
- Reduce job used = 0
- AVG=456,957,952

Virtual memory (bytes) snapshot:

- Map job used = 1,495,048,192
- Reduce job used = 0
- AVG= 1,495,048,192

5.2.3 RESULTS OF “select” QUERY IN SPARK CLUSTER

These results have get from “SparkJobs” interface, which shows the job id, stages, submitted time, and job duration, as shown in figure (5.2).

Spark Jobs (?)

User: hdusr
Total Uptime: 5.3 min
Scheduling Mode: FIFO
Completed Jobs: 1

▶ Event Timeline

Completed Jobs (1)

Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
0	show at <console>.29	2017/10/17 14:55:57	5 s	1/1	1/1

Figure 5.2: result measurements in “SparkJobs”

We could not get performance measurements of machines in spark, because “sparkJobs” interface does not show them, and we did not find another interface that show these measurements.

5.2.4 COMPARISON RESULT BASED ON TIME OF PROCESSING

Date records	420,000,000
Framework	
Hadoop MapReduce	1mins, 42sec
Spark	5 Sec

Table 2: Table 5.1 illustrate comparison result

According to this table:

□When implementing “select” query job, we found that “Hadoop MapReduce” takes a lot of time to do this job, but spark gives you the result per “Enter press”, as you see above by numbers. Although this is very simple job but Hadoop MapReduce takes time greater than spark time, it takes more than doubled time compare with spark. This result is because Spark do its processing and store part of input data in memory not in disk, but Hadoop MapReduce do it’s processing in disk, and absolutely access to memory is very fast than access to disk.

5.2.5 COMPARISON RESULTS

- As a comparison between Hadoop ecosystem configuration and spark configuration; we found that Hadoop has very *complex configuration*, but “Spark” has *simple and clear configuration*.
- “Hadoop” has *clear* error messages, and problems caused on it can be understood easily, but spark gives unclear exceptions, we took a lot of time to understand cause of problem or exceptions.
- Hadoop MapReduce has great support on the internet, many communities show questions and answers which help us very much, but Spark has poor support over the internet, because Spark is new technology which appear in 2014.

5.3 CONCLUSION:

From the experimental work, we found Spark overcomes Hadoop MapReduce performance in all cases. We conclude that several factors can give a rise to a significant performance difference. First, Spark pipelines resilient distributed datasets (RDDs) transform and keep persistent RDDs in memory by default, but Hadoop mainly concentrates on high throughput of data rather than on job execution performance, such that MapReduce results in overheads due to data replication, disk I/O, and serialization, which can dominate application execution times. Finally, yet importantly, Spark has more optimizations, such as the number of disk accesses per second, memory bandwidth utilization and IPC rate, than Hadoop, so that it provides a better performance, Spark is sure to be the best fit.

5.4 RECOMMENDATIONS

- We recommend to do the “select” query job multiple times in “Spark” and “Hadoop MapReduce”, and each time of running, increase one of machine resources such as, CPU cores, RAM used, or Disk space, to find that which resource affect the job performance in, and in which case “Spark” has better performance than “Hadoop MapReduce” and the opposite also.
- Use more complex query rather than select query, when the machines has big disk space and large memory.
- Design an intelligent system that can help to choose a platform and the configuration parameters, based on the applications and the input data sizes to get the optimized performance.

References

- [1] UK , Fujitsu ; , Irelan;, BigDataThe definitive guide to therevolution in business analytics, 2012.
- [2] Bhadani, Abhay Kumar ; Jothimani , Dhanya ;, "Big Data: Challenges, Opportunities and Realities," in *Effective Big Data Management and Opportunities for*, India, Indian Institute of Technology Delhi, 2015.
- [3] "Challenges and Opportunities with Challenges BigData," A community white paper developed by leading researchers across the United States, 2015.
- [4] . N. Samal and N. Mishra, "Big Data Processing: Big Challenges and Opportunities," *Journal of Computer Sciences and Applications*, vol. 3, pp. 177-180, 2015.
- [5] . M. Mayo and . K. , "Top Big Data Processing Frameworks," 2016. [Online]. Available: <https://www.kdnuggets.com/2016/03/top-big-data-processing-frameworks.html>. [Accessed 16 10 2017].
- [6] Elmasri, Ramez ; Navathe, Shamkant B;, *FUNDAMENTALS OF Database Systems SEVENTH EDITION*, Texas : Department of Computer Science and EngineeringThe University of Texas at Arlington, 2016, 2011, 2007.
- [7] Liu; Lu;, "Performance comparison by running benchmarks on Hadoop, Spark, and HAMR," no. 2015, 2016.
- [8] C. Hope, "Batch processing," 10 2 2017. [Online]. Available: <https://www.computerhope.com/jargon/b/batchpro.htm>. [Accessed 17 10 2017].
- [9] S. Shahrivari, "Beyond Batch Processing: Towards Real-Time and Streaming Big Data," *computers*, 2014.
- [10] T. A. S. Foundation., "Welcome to Apache™ Hadoop®!", 4 10 2017. [Online]. Available: <http://hadoop.apache.org/>. [Accessed 18 10 2017].
- [11] A. S. Foundation., "Apache Storm," [Online]. Available: <http://storm.apache.org/>. [Accessed 18 10 2017].

- [12] T. A. S. Foundation, "Apache Flink® is an open-source stream processing framework for distributed, high-performing, always-available, and accurate data streaming applications," [Online]. Available: <https://flink.apache.org/>. [Accessed 18 10 2017].
- [13] S , Sunil Kumar ; G , Sanjeev Kanabargji, "Challenges for HDFS to Read and Write Using Different Technologies," *International Journal of Science and Research (IJSR)* , pp. 1-6, 2013.
- [14] Armbrust, Michael; Das, Tathagata; Davidson, Aaron ; Ghodsi, Ali ; Or, Andrew ; Rosen, Josh ; Stoica, Ion; Wendell, Patrick ; Xin, Reynold ; Zaharia, Matei ;, "Scaling Spark in the Real World," *Proceedings of the VLDB Endowment - Proceedings of the 41st International Conference on Very Large Data Bases, Kohala Coast, Hawaii*, vol. 8, no. 12, pp. 1840-1843 , 2015.
- [15] Kevin , Jacobs; Kacper, Surdy; CERN. Geneva. IT Department;, "Apache Flink: Distributed Stream Data Processing," 2016.
- [16] Francis, Navya ; K, Sheena Kurian ;, "Data Processing for Big Data Applications," *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 4, no. 8, 2015.
- [17] Kaur, Dilraj ; Chadha, Raman ; Verma, Nitin, "COMPARISON OF MICRO - BATCH AND STREAMING ENGINE ON REAL TIME DATA," *INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCHTECHNOLOGY*, no. 2277-9655, 2017.
- [18] R. Sliberman, "A benchmarking tool for Streaming systems," *Big Data Architect*, 2015.
- [19] D. Borthakur, "The Hadoop Distributed File System:Architecture and Design.," The Apache Software Foundation, 2007.
- [20] [Online]. Available: <http://spark.apache.org/>.
- [21] "Apache Spark Primer," in *databricks*, 2017.
- [22] [Online]. Available:
<http://spark.apache.org/docs/0.8.1/api/core/org/Apache/spark/rdd/RDD.html>.
- [23] "Apache Hive," Apache Community Development Project., 2017. [Online]. Available:
<http://projects.apache.org/project.html?hive>.

- [24] "StackOverFlow," 12 January 2016. [Online]. Available: <https://stackoverflow.com/questions/13911501/when-to-use-hadoop-hbase-hive-and-pig>. [Accessed 19 10 2017].
- [25] Prof; Zhan, Jianfeng ;, BigDataBench User Manual, ICT, Chinese Academy of Sciences.
- [26] C. A. o. S. ICT, "BigDataBench," 16 February 2017. [Online]. Available: <http://prof.ict.ac.cn/#WhyBigDataBench>. [Accessed 19 10 2017].
- [27] [Online]. Available: <https://www.ibm.com/analytics/us/en/technology/Hadoop/MapReduce/>.
- [28] Office of the Vice President for Management and Budget University of Virginia , "Benchmarking in higher education".

APPENDICES

APPENDIX (A)

CONFIGURATION OF FRAMEWORKS

INTRODUCTION

To insure that the frameworks or the environment working well, we introduce the steps of Hadoop , Spark and BigBench installations and configurations and also we show some snapshots of the important steps that shows result when executed.

HADOOP CONFIGURATION

Before install Hadoop framework we have Prerequisite:

First, you must install ssh server:

* SSH, or *Secure Shell*, is a protocol used to securely log onto remote systems.

It is the most common way to access remote Linux and Unix-like servers.

Fire this command to do this:

```
$ sudo apt-get install openssh-server
```

then to make your machine communicate with one another without any prompt for password:

this command to generate key between machines via:

```
$ ssh-keygen -t rsa this
```

command to copy the id via:

```
$ ssh-copy-id -i ~/.ssh/id_rsa.pub ubuntu@master this
```

command to change the mod:

```
$ chmod 0600 ~/.ssh/authorized_keys then
```

reboot or restart the machine:

```
$ sudo reboot
```

Second, install java:

*Java is the main prerequisite for Hadoop, here We install java 7. first

check if Java is not already installed, fire this:

\$ java -version if it not

exist install it via:

```
$ sudo apt-get install default-jre
```

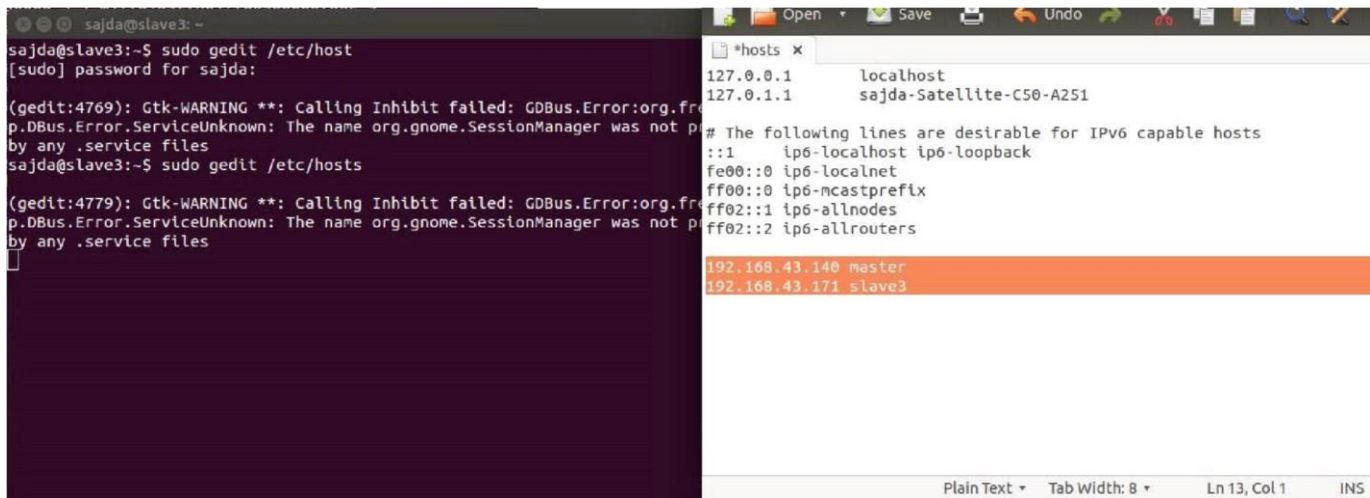
```
$ sudo apt-get install default-jdk
```

Thirdly, download and install Apache Hadoop source file, using the following commands.:

*First thing you must sure that the ip address of master and slaves write in the /etc/hosts via:

```
$ sudo gedit /etc/hosts
```

Checking for the IP address.



*Download Hadoop From

```
http://www-eu.apache.org/dist/Hadoop/common/
```

*here, you can find the latest version and latest modification.

*we download Hadoop-2.7.0.tar.gz and extract it in /usr/local via this command:

```
$ sudo tar -xvf Hadoop-2.7.0.tar.gz
```

Extract Hadoop file.

```
sajda@slave3: /usr/local
hadoop-2.7.0/libexec/yarn-config.cmd
hadoop-2.7.0/libexec/hadoop-config.cmd
hadoop-2.7.0/libexec/mapred-config.sh
hadoop-2.7.0/libexec/httpfs-config.sh
hadoop-2.7.0/libexec/hadoop-config.sh
hadoop-2.7.0/libexec/mapred-config.cmd
hadoop-2.7.0/libexec/kms-config.sh
hadoop-2.7.0/libexec/hdfs-config.cmd
hadoop-2.7.0/libexec/yarn-config.sh
hadoop-2.7.0/libexec/hdfs-config.sh
hadoop-2.7.0/README.txt
hadoop-2.7.0/NOTICE.txt
hadoop-2.7.0/lib/
hadoop-2.7.0/lib/native/
hadoop-2.7.0/lib/native/libhadoop.a
hadoop-2.7.0/lib/native/libhadoop.so
hadoop-2.7.0/lib/native/libhadooppipes.a
hadoop-2.7.0/lib/native/libhdfs.so.0.0.0
hadoop-2.7.0/lib/native/libhadooputils.a
hadoop-2.7.0/lib/native/libhdfs.a
hadoop-2.7.0/lib/native/libhdfs.so
hadoop-2.7.0/lib/native/libhadoop.so.1.0.0
hadoop-2.7.0/LICENSE.txt
sajda@slave3: /usr/local$
```

- Fourthly, then give the Hadoop file permission and privileges after you download it via these steps:

```
$ sudo chown ubuntu:root /usr/local/Hadoop-2.7.0
```

```
$ sudo chmod 777 /usr/local/Hadoop-2.7.0/*
```

*ubuntu: is the user name in you machine.

*root: is group that had the permission and privilege.

- Fifthly, create Hadoop_tmp/hdfs :

```
$ sudo mkdir -p /usr/local/Hadoop_tmp/hdfs
```

* And then give it permission and privileges via:

```
$ sudo chown ubuntu:root /usr/local/Hadoop_tmp
```

```
$ sudo chmod 777 /usr/local/Hadoop_tmp/*
```

- We need to Update .bashrc file :

*.bashrc is a shell script that Bash runs whenever it is started interactively. You can put any command in that file that you could type at the command prompt. You put commands here to set up the shell for use in your particular environment, or to customize things to your preferences.

- Fire via:

```
$ sudo gedit .bashrc
```

- in file .bashrc we write the java home and Hadoop home in the end of the file :

```
# -- HADOOP ENVIRONMENT VARIABLES START -- #
```

```
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64
```

```
export HADOOP_HOME=/usr/local/Hadoop-2.7.0 export
```

```
PATH=$PATH:$HADOOP_HOME/bin export
```

```
PATH=$PATH:$HADOOP_HOME/sbin export
```

```
HADOOP_MAPRED_HOME=$HADOOP_HOME export
```

```
HADOOP_COMMON_HOME=$HADOOP_HOME
```

```
export HADOOP_HDFS_HOME=$HADOOP_HOME
```

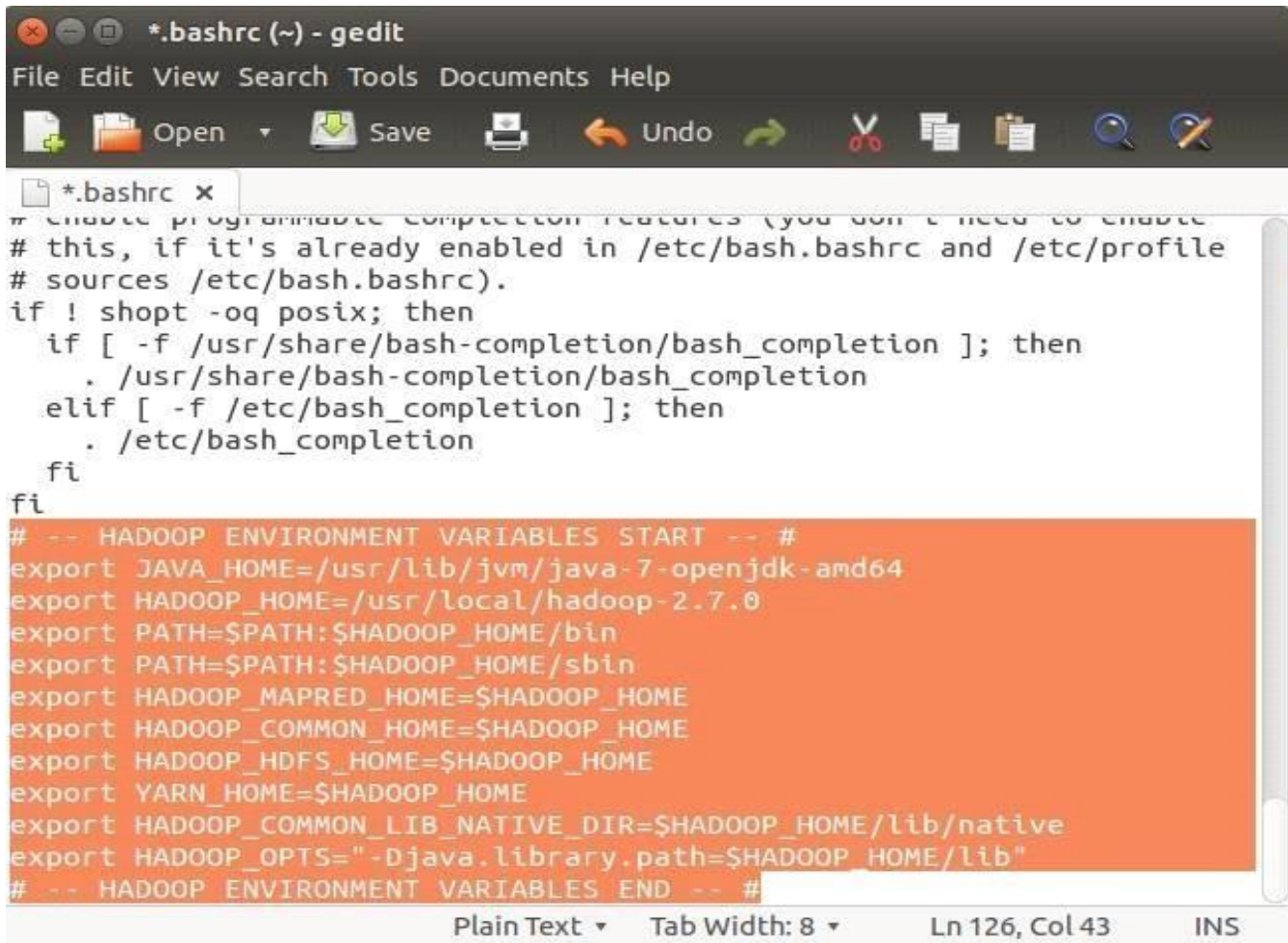
```
export YARN_HOME=$HADOOP_HOME
```

```
export
```

```
HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
```

```
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib"
```


-- HADOOP ENVIRONMENT VARIABLES END --



```
*.bashrc (~) - gedit
File Edit View Search Tools Documents Help
Open Save Undo
*.bashrc x
# Enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi
# -- HADOOP ENVIRONMENT VARIABLES START -- #
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64
export HADOOP_HOME=/usr/local/hadoop-2.7.0
export PATH=$PATH:$HADOOP_HOME/bin
export PATH=$PATH:$HADOOP_HOME/sbin
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib"
# -- HADOOP ENVIRONMENT VARIABLES END -- #
Plain Text Tab Width: 8 Ln 126, Col 43 INS
```

Write java path in .bashrc file.

□ Sixthly, Configuration files :

1. Hadoop-env.sh its location in we Update JAVA_HOME variable:

/usr/local/Hadoop-2.7.0/etc/Hadoop

*fire this command:

\$ sudo gedit Hadoop-env.sh

*and write:

JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64

2. core-site.xml:

```
$ sudo gedit core-site.xml
```

*Paste these lines into <configuration> tag

```
<property>
```

```
<name>fs.default.name</name>
```

```
<value>hdfs://localhost:9000</value>
```

```
</property>
```

3. hdfs-site.xml:

```
$ sudo gedit hdfs-site.xml
```

*Paste these lines into <configuration> tag

```
<property>
```

```
<name>dfs.replication</name>
```

```
<value>1</value>
```

```
</property>
```

```
<property>
```

```
<name>dfs.namenode.name.dir</name>
```

```
<value>/usr/local/Hadoop_tmp/hdfs/namenode</value>
```

```
</property>
```

```
<property>
```

```
<name>dfs.datanode.data.dir</name>
```

```
<value>/usr/local/Hadoop_tmp/hdfs/datanode</value> </property>
```

4. yarn-site.xml:

```
$ sudo gedit yarn-site.xml
```

*Paste these lines into <configuration> tag

```
<property>
```

```
<name>yarn.nodemanager.aux-services</name>
```

```
<value>MapReduce_shuffle</value>
```

```
</property>
```

```
<property>
```

```
<name>yarn.nodemanager.auxservices.MapReduce.shuffle.class</name>
```

```
<value>org.Apache.Hadoop.mapred.ShuffleHandler</value>
```

```
</property>
```

5. mapred-site.xml:

*Copy template of mapred-site.xml.template file via:

```
$ sudo cp /usr/local/Hadoop-2.7.0/etc/Hadoop/mapred-site.xml.template  
/usr/local/Hadoop-2.7.0/etc/Hadoop/mapred-site.xml
```

*To edit file, fire the below given command

```
$ sudo gedit mapred-site.xml
```

*Paste these lines into <configuration> tag

```
<property>
```

```
<name>MapReduce.framework.name</name>
```

```
<value>yarn</value>
```

</property>

*Format Namenode in Hadoop-2.7.0 master:

*Hadoop NameNode is the centralized place of an HDFS file system which keeps the directory tree of all files in the file system, and tracks where across the cluster the file data is kept. In short, it keeps the metadata related to datanodes. When we format namenode it formats the meta-data related to data-nodes. By doing that, all the information on the datanodes are lost and they can be reused for new data.

* fire via:

\$ bin/Hadoop namenode -format

Format namenode



```
sajda@slave3: /usr/local/hadoop-2.7.0
sajda@slave3: /usr/local/hadoop-2.7.0$ bin/hadoop namenode -format
```

```

sajda@slave3: /usr/local/hadoop-2.7.0
17/08/02 15:58:45 INFO metrics.TopMetrics: NNTop conf: dfs.namenode.top.num.users = 10
17/08/02 15:58:45 INFO metrics.TopMetrics: NNTop conf: dfs.namenode.top.windows.minutes = 1,5,25
17/08/02 15:58:45 INFO namenode.FSNamesystem: Retry cache on namenode is enabled
17/08/02 15:58:45 INFO namenode.FSNamesystem: Retry cache will use 0.03 of total heap and retry cache entry expiry time is 600000 millis
17/08/02 15:58:45 INFO util.GSet: Computing capacity for map NameNodeRetryCache
17/08/02 15:58:45 INFO util.GSet: VM type = 64-bit
17/08/02 15:58:45 INFO util.GSet: 0.029999999329447746% max memory 889 MB = 273.1 KB
17/08/02 15:58:45 INFO util.GSet: capacity = 2^15 = 32768 entries
17/08/02 15:58:45 INFO namenode.FSImage: Allocated new BlockPoolId: BP-758996794-192.168.43.171-1501678725047
17/08/02 15:58:45 INFO common.Storage: Storage directory /usr/local/hadoop_tmp/hdfs/namenode has been successfully formatted.
17/08/02 15:58:45 INFO namenode.NNStorageRetentionManager: Going to retain 1 images with txid >= 0
17/08/02 15:58:45 INFO util.ExitUtil: Exiting with status 0
17/08/02 15:58:45 INFO namenode.NameNode: SHUTDOWN_MSG:
/*****
SHUTDOWN_MSG: Shutting down NameNode at slave3/192.168.43.171
*****/
sajda@slave3: /usr/local/hadoop-2.7.0$

```

Formatted namenode successfully

- if namenode don't started in master:
- you must sure that if the Hadoop_tmp that contain namenode and datanode :
- Delete and recreate Hadoop_tmp/hdfs :

```
$ sudo rm -rf /usr/local/Hadoop_tmp $ sudo
```

```
mkdir -p /usr/local/Hadoop_tmp/hdfs
```

- have the Hadoop user owner:

```
$ sudo chown ubuntu:root /usr/local/Hadoop_tmp
```

- give privileges:

```
$ sudo chmod 777 /usr/local/Hadoop_tmp/*
```

- and then:
- Start all Hadoop daemons :
 - * Used to start and stop Hadoop daemons all at once. Issuing it on the master

machine will start/stop the daemons on all the nodes of a cluster. Deprecated as you have already noticed.

- Fire via:

\$ sbin/start-all.sh

```
sajda@slave3: /usr/local/hadoop-2.7.0
sajda@slave3:/usr/local/hadoop-2.7.0$ sbin/start-all.sh
This script is Deprecated. Instead use start-dfs.sh and start-yarn.sh
17/08/02 16:21:01 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Starting namenodes on [slave3]
slave3: starting namenode, logging to /usr/local/hadoop-2.7.0/logs/hadoop-sajda-namenode-slave3.out
slave3: starting datanode, logging to /usr/local/hadoop-2.7.0/logs/hadoop-sajda-datanode-slave3.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: secondarynamenode running as process 2770. Stop it first.
17/08/02 16:21:15 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
starting yarn daemons
resourcemanager running as process 2968. Stop it first.
slave3: nodemanager running as process 3098. Stop it first.
sajda@slave3:/usr/local/hadoop-2.7.0$ jpsa
No command 'jpsa' found, did you mean:
  Command 'jps' from package 'openjdk-7-jdk' (main)
  Command 'jps' from package 'openjdk-6-jdk' (universe)
  Command 'jp2a' from package 'jp2a' (universe)
jpsa: command not found
sajda@slave3:/usr/local/hadoop-2.7.0$ spd
The program 'spd' is currently not installed. You can install it by typing:
```

Start Hadoop daemons

- Last thing to grantee you work is done correctly fire jps:

\$ jps

- That must show:

jps to show the daemons of Hadoop

```
sajda@slave3: /usr/local/hadoop-2.7.0
sajda@slave3:/usr/local/hadoop-2.7.0$ jps
3778 NameNode
3098 NodeManager
2968 ResourceManager
3937 DataNode
4698 Jps
2770 SecondaryNameNode
sajda@slave3: /usr/local/hadoop-2.7.0$
```

*then open the namenode site:

localhost:50070

The screenshot shows a web browser window displaying the Hadoop NameNode overview page. The browser address bar shows 'localhost:50070/dfshealth.html#tab-overview'. The page has a green navigation bar with tabs for 'Hadoop', 'Overview', 'Datanodes', 'Datanode Volume Failures', 'Snapshot', 'Startup Progress', and 'Utilities'. The main content area is titled 'Overview 'localhost:9000' (active)'. Below the title is a table with the following information:

Started:	Tue Aug 15 17:44:04 +0300 2017
Version:	2.8.1, r20fe5304904fc2f5a18053c389e43cd26f7a70fe
Compiled:	Fri Jun 02 09:14:00 +0300 2017 by vinodkv from branch-2.8.1-private
Cluster ID:	CID-40728cb1-4ff1-41b8-bec4-31e2bb804578
Block Pool ID:	BP-114661778-192.168.43.8-1502808219083

Below the table is a 'Summary' section with the following text:

Security is off.
 Safemode is off.
 \ files and directories, - blocks = \ total filesystem object(s).

At the bottom of the page, there is a 'Configured Capacity:' label and a value of '93.79 GB'. A video player overlay is visible at the bottom of the screenshot, showing a play button, a progress bar at 15:46, and a volume icon.

Namenode site

The screenshot shows the Hadoop YARN web interface at localhost:8088/cluster. The page title is 'All Applications'. On the left, there is a navigation menu with options like 'Cluster', 'About Nodes', 'Node Labels', 'Applications', and 'Scheduler'. The main content area is divided into several sections: 'Cluster Metrics' (a table with columns for Apps Submitted, Pending, Running, Completed, Containers Running, Memory Used, Total, Reserved, and V-Cores Used), 'Cluster Nodes Metrics' (a table with columns for Active, Decommissioning, Decommissioned, Lost, Unhealthy, and Rebooted nodes), 'Scheduler Metrics' (a table with columns for Scheduler Type, Scheduling Resource Type, Minimum Allocation, Maximum Allocation, and Maximum Capacity), and a table for 'Applications' (currently empty). The status bar at the bottom indicates 'Showing 0 to 0 of 0 entries'.

and the yarn site:

localhost:8088

Yarn site

SPARK CONFIGURATION

Before install Spark framework we have Prerequisite:

- First, like Hadoop framework we must do the same command before install Spark framework ,we must install ssh server via following commands:

\$ sudo apt-get install openssh-server

\$ ssh-keygen -t rsa

\$ ssh-copy-id -i ~/.ssh/id_rsa.pub ubuntu@master

\$ chmod 0600 ~/.ssh/authorized_keys

- Then reboot the machine:

\$ sudo reboot

- Secondly, install java:
- first check if Java is not already installed:

\$ java -version

- if it not exist install it via:

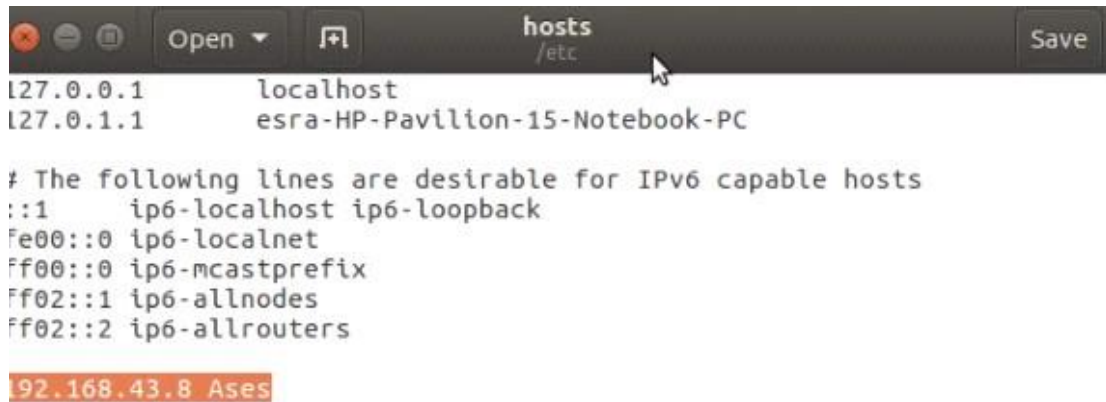
\$ sudo apt-get install default-jre \$

sudo apt-get install default-jdk

- Thirdly, download Spark via these steps:

*First thing you must sure that the IP address of master and slaves write in the /etc/hosts via:

\$ sudo gedit /etc/hosts



```
hosts
/etc
127.0.0.1    localhost
127.0.1.1    esra-HP-Pavilion-15-Notebook-PC

# The following lines are desirable for IPv6 capable hosts
::1        ip6-localhost ip6-loopback
fe00::0    ip6-localnet
ff00::0    ip6-mcastprefix
ff02::1    ip6-allnodes
ff02::2    ip6-allrouters

192.168.43.8 Ases
```



Checking for the IP address.

- In next step install git, Spark build depends on git:

```
$ sudo apt-get install git
```

- Fourthly, in next step is install Scala, follow the following instructions to set up Scala. First download the Scala from:

<https://scala-lang.org/download/>

- Here we install scala-2.12.3.tgz □ Extract it via:

```
$ sudo tar -xvf scala-2.12.3.tgz
```

- And then give it permission and privilege via:

```
$ sudo chown ubuntu:root /usr/local/scala-2.12.3
```

```
$ sudo chmod 777 /usr/local/scala-2.12.3/*
```

```
esra@Ases: /usr/local
scala-2.12.3/doc/
scala-2.12.3/doc/tools/
scala-2.12.3/doc/tools/scala.html
scala-2.12.3/doc/tools/index.html
scala-2.12.3/doc/tools/images/
scala-2.12.3/doc/tools/images/scala_logo.png
scala-2.12.3/doc/tools/images/external.gif
scala-2.12.3/doc/tools/fsc.html
scala-2.12.3/doc/tools/scalac.html
scala-2.12.3/doc/tools/css/
scala-2.12.3/doc/tools/css/style.css
scala-2.12.3/doc/tools/scalap.html
scala-2.12.3/doc/tools/scaladoc.html
scala-2.12.3/doc/License.rtf
scala-2.12.3/doc/licenses/
scala-2.12.3/doc/licenses/bsd_asm.txt
scala-2.12.3/doc/licenses/mit_jquery.txt
scala-2.12.3/doc/licenses/bsd_jline.txt
scala-2.12.3/doc/licenses/mit_sizzle.txt
scala-2.12.3/doc/licenses/mit_tools_tooltip.txt
scala-2.12.3/doc/licenses/apache_jansi.txt
scala-2.12.3/doc/LICENSE.md
scala-2.12.3/doc/README
esra@Ases: /usr/local$
```

Extract Scala file.

- In the next step, install the source of Apache Spark from:

<https://Spark.apache.org/downloads.html>

- Here we install Spark-2.2.0 □ Extract it via:

```
$ sudo tar -xvf Spark-2.2.0-bin-Hadoop2.7.tgz
```

- And then give it permission and privilege via:

```
$ sudo chown ubuntu:root /usr/local/Spark-2.2.0-bin-  
Hadoop2.7
```

```
$ sudo chmod 777 /usr/local/Spark-2.2.0-bin-Hadoop2.7/*
```

```
esra@Ases: /usr/local
spark-2.2.0-bin-hadoop2.7/bin/pyspark
spark-2.2.0-bin-hadoop2.7/bin/sparkR.cmd
spark-2.2.0-bin-hadoop2.7/bin/spark-class2.cmd
spark-2.2.0-bin-hadoop2.7/bin/run-example.cmd
spark-2.2.0-bin-hadoop2.7/bin/spark-submit2.cmd
spark-2.2.0-bin-hadoop2.7/bin/spark-class
spark-2.2.0-bin-hadoop2.7/bin/spark-submit
spark-2.2.0-bin-hadoop2.7/bin/spark-sql
spark-2.2.0-bin-hadoop2.7/bin/find-spark-home
spark-2.2.0-bin-hadoop2.7/bin/run-example
spark-2.2.0-bin-hadoop2.7/bin/beeline
spark-2.2.0-bin-hadoop2.7/bin/pyspark2.cmd
spark-2.2.0-bin-hadoop2.7/bin/spark-shell.cmd
spark-2.2.0-bin-hadoop2.7/bin/spark-class.cmd
spark-2.2.0-bin-hadoop2.7/bin/pyspark.cmd
spark-2.2.0-bin-hadoop2.7/bin/sparkR
spark-2.2.0-bin-hadoop2.7/bin/beeline.cmd
spark-2.2.0-bin-hadoop2.7/bin/sparkR2.cmd
spark-2.2.0-bin-hadoop2.7/bin/load-spark-env.sh
spark-2.2.0-bin-hadoop2.7/bin/load-spark-env.cmd
spark-2.2.0-bin-hadoop2.7/yarn/
spark-2.2.0-bin-hadoop2.7/yarn/spark-2.2.0-yarn-shuffle.jar
spark-2.2.0-bin-hadoop2.7/README.md
esra@Ases: /usr/local$
```

Extract Spark file.

- Then configure these Files:
- Edit the .bashrc file by write:

```
$ sudo gedit .bashrc
```

- And write in it:
- the Scala home:

```
export SCALA_HOME=/usr/local/scala-2.12.3 export
```

```
PATH=$SCALA_HOME/bin:$PATH
```

- And Spark home:

```
export SPARK_HOME=/usr/local/Spark-2.2.0-bin-Hadoop2.7
```

```
export PATH=$SPARK_HOME/bin:$PATH
```

```
fi
fi

# -- SPARK ENVIRONMENT VARIABLES START -- #
export SCALA_HOME=/usr/local/scala-2.12.3
export PATH=$SCALA_HOME/bin:$PATH

#the spark home:
export SPARK_HOME=/usr/local/spark-2.2.0-bin-hadoop2.7
export PATH=$SPARK_HOME/bin:$PATH

# -- SPARK ENVIRONMENT VARIABLES END -- #

# -- HADOOP ENVIRONMENT VARIABLES START -- #
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
export HADOOP_HOME=/usr/local/hadoop-2.8.1
export PATH=$PATH:$HADOOP_HOME/bin
export PATH=$PATH:$HADOOP_HOME/sbin
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
```

Edit .bashrc file.

- Copy the following files, and can find it in /Spark/conf/:
- fire these command:

1. slaves file:

```
$ cp slaves.template slaves
```

```
$ sudo gedit slaves
```

- And then type inside file :

```
localhost
```

```
# Licensed to the Apache Software Foundation (ASF) under one or more
# contributor license agreements.  See the NOTICE file distributed
# with
# this work for additional information regarding copyright ownership.
# The ASF licenses this file to You under the Apache License,
# Version 2.0
# (the "License"); you may not use this file except in compliance
# with
# the License.  You may obtain a copy of the License at
#
#   http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
# implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#
# A Spark Worker will be started on each of the machines listed
# below.
localhost
```

The slaves file.

2. Spark-default.conf file:

```
$ cp Spark-defaults.conf.template Spark-defaults.conf
```

```
$ sudo gedit Spark-defaults.conf
```

- And then type inside file :

```
Spark.master Spark://localhost:7077
```



```
wcch
# the License. You may obtain a copy of the License at
#
#   http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
# implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#

# Default system properties included when running spark-submit.
# This is useful for setting default environmental settings.

# Example:
# spark.master                spark://master:7077
# spark.eventLog.enabled      true
# spark.eventLog.dir          hdfs://namenode:8021/directory
# spark.serializer            org.apache.spark.serializer.KryoSerializer
# spark.driver.memory          5g
# spark.executor.extraJavaOptions -XX:+PrintGCDetails -Dkey=value -
#                               Dnumbers="one two three"
spark.master spark://localhost:7077

Saving file '/usr/loca... Plain Text ▾ Tab Width: 8 ▾ Ln 28, Col 36 ▾ INS
```

The Spark-default.conf file.

3. Spark-env.sh file:

```
$ cp Spark-env.sh.template Spark-env.sh
```

```
$ sudo gedit Spark-env.sh
```

- And then type inside file :

```
export SCALA_HOME=/usr/local/scala-2.12.3
```

```
*spark-env.sh
/usr/local/spark-2.2.0-bin-hadoop2.7/conf
# - SPARK_WORKER_OPTS, to set config properties only for the worker
(e.g. "-Dx=y")
# - SPARK_DAEMON_MEMORY, to allocate to the master, worker and
history server themselves (default: 1g).
# - SPARK_HISTORY_OPTS, to set config properties only for the
history server (e.g. "-Dx=y")
# - SPARK_SHUFFLE_OPTS, to set config properties only for the
external shuffle service (e.g. "-Dx=y")
# - SPARK_DAEMON_JAVA_OPTS, to set config properties for all daemons
(e.g. "-Dx=y")
# - SPARK_PUBLIC_DNS, to set the public dns name of the master or
workers

# Generic options for the daemons used in the standalone deploy mode
# - SPARK_CONF_DIR      Alternate conf dir. (Default: ${SPARK_HOME}/
conf)
# - SPARK_LOG_DIR       Where log files are stored. (Default:
${SPARK_HOME}/logs)
# - SPARK_PID_DIR       Where the pid file is stored. (Default: /tmp)
# - SPARK_IDENT_STRING  A string representing this instance of
spark. (Default: $USER)
# - SPARK_NICENESS      The scheduling priority for daemons.
(Default: 0)
# - SPARK_NO_DAEMONIZE  Run the proposed command in the foreground.
It will not output a PID file.
export SCALA_HOME=/usr/local/scala-2.12.3|
sh Tab Width: 8 Ln 63, Col 42 INS
```

The Spark-env.sh file.

- Lastly start all Spark daemons:

```
$ sbin/start-master.sh
```

```
$ sbin/start-slaves.sh
```

- if these command not run, fire this:

```
$ sbin/start-all.sh
```



```
esra@Ases: /usr/local/spark-2.2.0-bin-hadoop2.7
esra@Ases:/usr/local/spark-2.2.0-bin-hadoop2.7$ sbin/start-master.sh
starting org.apache.spark.deploy.master.Master, logging to /usr/local/spark-2.2.0-bin-hadoop2.7/logs/spark-esra-org.apache.spark.deploy.master.Master-1-Ases.out
esra@Ases:/usr/local/spark-2.2.0-bin-hadoop2.7$ sbin/start-slaves.sh
localhost: starting org.apache.spark.deploy.worker.Worker, logging to /usr/local/spark-2.2.0-bin-hadoop2.7/logs/spark-esra-org.apache.spark.deploy.worker.Worker-1-Ases.out
esra@Ases:/usr/local/spark-2.2.0-bin-hadoop2.7$
```

start all Spark daemons.

- To guarantee that was working type :

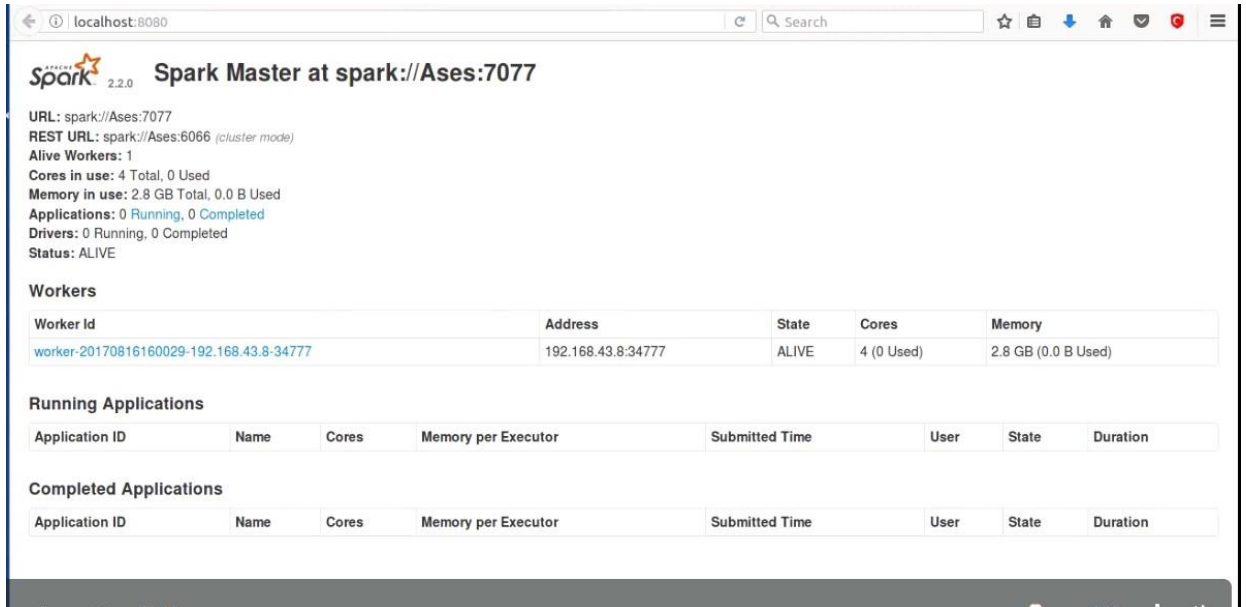
\$ jps

```
esra@Ases:/usr/local/spark-2.2.0-bin-hadoop2.7$ jps
9554 Jps
9428 Worker
9295 Master
esra@Ases:/usr/local/spark-2.2.0-bin-hadoop2.7$
```

jps to show the daemons of Spark

- open Spark sites:

localhost:8080

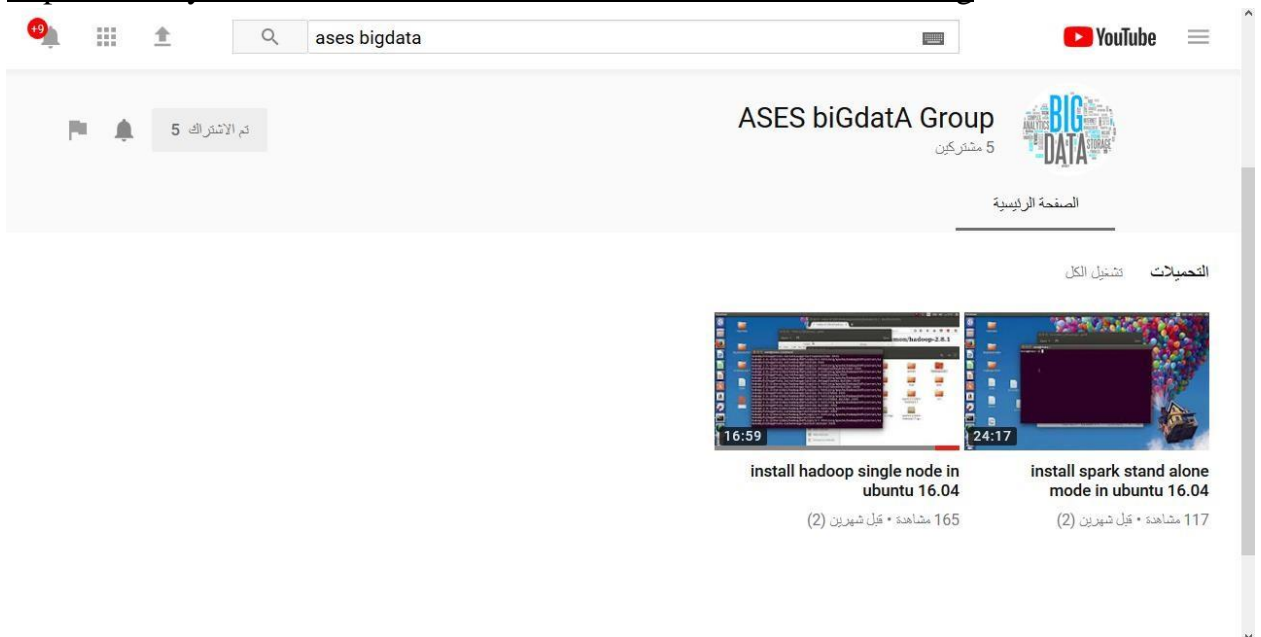


Spark website

HINT:

To clarify more, we have done work a channel with videos explaining for frameworks configuration can found it in this link:

<https://www.youtube.com/channel/UCW6umaU5sDzBXTf-vRUDeIg>



APPINDEX (B)

BIG DATA IN SUDAN QUESIONNAIRE

INTRODUCTION:

Big data in Sudan survey is a survey which targeted many companies in many sectors in Sudan.

SURVEY QUESTIONS:

Company Name?

.....

Employee name?

.....

Department name?

.....

Did your company has big data (e.g Billion record or 1 Tera volume of data)?

- Y
- N

Did this big data make latency in company's decision(e.g delaying report or delaying decision ...etc)

- Y
- N

Which techniques do you use to process this data?

.....
Did you use Batch processing?

es • Y

o • N

Did you use parallel processing?

es • Y

o • N

Where you store this big data?

istributed • D

entral • C

Did you store data in HDFS?

es • Y

o • N

Did you use Hadoop MapReduce to implement this big data?

es • Y

o • N

Did you hear about Apache Spark?

es • Y

- N

o

Did you use Apache Spark to implement this big data?

- Y

es

- N

o **HINT:**

THIS IS THE SITE OF SURVEY:

<https://docs.google.com/forms/d/e/1FAIpQLSfTCtxCK4xATXxC58wNY55LI33mB8apI1BKjwkt6NocWcLDA/viewform>